INCREMENTAL REFRESH MATERIALIZED QUERY TABLE(MQT) MEMANFAATKAN STAGING TABLE UNTUK OPTIMASI QUERY EXECUTION TIME DAN RESOURCES YANG DIGUNAKAN

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Dirga Harjuna Putra
NIM: 115061000111033

PROGRAM STUDI SISTEM INFORMASI
JURUSAN SISTEM INFORMASI
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

INCREMENTAL REFRESH MATERIALIZED QUERY TABLE(MQT) MEMANFAATKAN STAGING TABLE UNTUK OPTIMASI QUERY EXECUTION TIME DAN RESOURCES YANG DIGUNAKAN

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Dirga Harjuna Putra
NIM: 115061000111033

Skripsi ini telah diuji dan dinyatakan lulus pada
31 Juli 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Aryo Pinandito, S.T, M.MT.	Djoko Pramono, S.T., M.Kom.
NIP: 198305192014041001	NIP: 197801082005011002

Mengetahui
Ketua Jurusan Sistem Informasi

Herman Tolle, Dr. Eng., S.T, M.T.
NIP: 197408232000121001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 15 Juli 2018

Dirga Harjuna Putra

NIM: 115061000111033

KATA PENGANTAR

Segala puji syukur penulis ucapkan kepada Allah SWT yang telah melimpahkan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan skripsi dengan judul "*Incremental Refresh Materialized Query Table* (MQT) Memanfaatkan Staging Table Untuk Optimasi Query Execution Time Dan Resources Yang Digunakan" sebagai salah satu syarat untuk memenuhi gelar Sarjana Komputer.

Dalam menyelesaikan skripsi ini penulis mendapat banyak dukungan, bantuan dan do'a dari banyak pihak. Oleh karena itu pada kesempatan ini penulis ingin menyampaikan rasa hormat dan terima kasih yang sebesar-besarnya kepada:

1. Bapak Aryo Pinandito, S.T, M.MT. selaku dosen pembimbing satu yang dengan sabar telah memberikan ilmu, arahan dan bimbingan.

2. Bapak Djoko Pramono, S.T., M.Kom. selaku dosen pembimbing satu yang dengan sabar telah memberikan ilmu, arahan dan bimbingan.

3. Seluruh tenaga pendidik yang telah mendidik dan memberikan banyak ilmu kepada penulis selama menempuh studi di Fakultas Ilmu Komputer.

4. Orangtua dan kedua adik saya yang telah mendukung, memberikan motivasi, do'a dan bantuan yang tiada henti-hentinya dalam menyelesaikan skripsi.

5. Keluarga besar Sistem Informasi Universitas Brawijaya serta teman-teman yang telah memberikan banyak pengalaman, ilmu dan dukungannya selama penulis menempuh studi di Fakultas Ilmu Komputer Universitas Brawijaya.

6. Seluruh teman-teman tenaga kependidikan di UPT Teknologi Informasi dan Komunikasi yang telah memberikan motivasi dan bantuan untuk menyelesaikan skripsi.

7. Semua pihak yang telah membantu pengerjaan skripsi, yang tidak dapat disebutkan satu persatu.

Semoga Allah SWT memberikan balasan kebaikan yang berlipat ganda bagi semuanya. Demi perbaikan selanjutnya, kritik dan saran yang membangun akan penulis terima dengan senang hati. Akhir kata penulis berharap skripsi ini dapat bermanfaat bagi semua pihak.

Malang, 15 Juli 2018

Penulis
dirgaharjuna@gmail.com

ABSTRAK

Materialized Query Table (MQT) menyimpan data dari *query* yang sering digunakan sehingga pengguna dapat memperoleh data tanpa harus melakukan komputasi ulang. Hal ini dapat meningkatkan performa sistem dengan mengurangi biaya *query*. Data didalam MQT harus diperbarui secara berkala agar tidak menjadi usang ketika terjadi perubahan pada tabel induk. Ada 2 (dua) macam mekanisme pembaruan yang umum digunakan, yaitu *full refresh* dan *incremental refresh*. *Full refresh* mengkomputasi ulang seluruh data dari tabel induk. Sedangkan *incremental refresh* hanya memproses data-data yang mengalami perubahan dengan memanfaatkan *staging table*. *Staging table* berperan menyimpan perubahan data (delta) pada tabel induk untuk mendukung proses *incremental refresh*.

Penelitian ini mensimulasikan dan membandingkan performa dari *full refresh* dengan *incremental refresh* untuk mengetahui dampak keduanya terhadap waktu eksekusi *query* dan penggunaan sumber daya (I/O dan CPU). Data yang digunakan merupakan data asli yang berasal dari penelitian sebelumnya dan data *dummy* hasil *generate* sistem untuk mendukung penelitian. Hasil pengujian menunjukkan bahwa *incremental refresh* meningkatkan performa lebih dari 10x lipat pada waktu eksekusi *query* dan meningkatkan lebih dari 50x lipat pada penggunaan sumber daya dibandingkan dengan *full refresh*.

Kata kunci: MQT, *staging table, refresh, full, incremental,* waktu eksekusi, I/O, CPU

ABSTRACT

Materialized Query Table (MQT) stores data from frequently used queries so that users can get data without having to re-compute. This can improve system performance by reducing the cost of queries. The data in the MQT must be updated periodically it will not become obsolete when some change being made in the parent table. There are 2 (two) kinds of commonly used update mechanisms, namely full refresh and incremental refresh. Full refresh recompiles all data from the parent table. While incremental refresh only process data that being changed by utilizing staging table. The staging table stores the data changes (delta) in the parent table so that incremental refresh can be performed.

This study simulates and compares the performance of full refresh with incremental refresh to know the impact of both of the query execution time and resource usage (I/O and CPU). The data used are original data derived from previous research and dummy data generated by system to support this research. Test results show that incremental refresh increases performance by more than 10 times at query execution time and increases more than 50 times on resource usage compared to full refresh.

Keywords: MQT, *staging table, refresh, full, incremental, execution time*, I/O, CPU

DAFTAR ISI

PENGESAHAN ii
PERNYATAAN ORISINALITAS iii
KATA PENGANTAR iv
ABSTRAK v
ABSTRACT vi
DAFTAR ISI vii
DAFTAR TABEL x
DAFTAR GAMBAR xi
DAFTAR LAMPIRAN xiv
BAB 1 PENDAHULUAN 1
1.1 Latar belakang 1
1.2 Rumusan masalah 3
1.3 Tujuan 3
1.4 Manfaat 3
1.5 Batasan masalah 4
1.6 Sistematika pembahasan 4
BAB 2 LANDASAN KEPUSTAKAAN 6
2.1 Kajian Pustaka 6
2.2 Database System 7
2.3 Database Management System 9
2.4 Aljabar Relasional 9
2.5 Temporal Data Management 11
2.5.1 Keuntungan *Temporal Data Management* 11
2.5.2 Manajemen Versi Data Menggunakan dengan *System Time* 12
2.6 Materialized Query Table 15
2.6.1 Keuntungan MQT 17
2.6.2 Jenis-Jenis MQT 18
2.6.3 Join Table Pada MQT 18
2.6.4 Mekanisme Pemutakhiran MQT 20

2.7 MQT Staging Table 21
2.7.1 Pembuatan Staging Table 22
2.7.2 Syarat dan Ketentuan Staging Table 22
2.8 Query Rewrite 23
BAB 3 METODOLOGI 25
3.1 Metode Penelitian 25
3.1.1 Studi Literatur 25
3.1.2 Pengumpulan Data Uji 26
3.1.3 Perancangan 26
3.1.4 Implementasi 27
3.1.5 Pengujian Optimasi Query 28
3.1.6 Penyusunan Laporan 28
BAB 4 PERANCANGAN 29
4.1 Perancangan Arsitektur 29
4.2 Perancangan Basis Data 29
4.3 Perancangan Data Penelitian 31
4.4 Perancangan Materialized Query Table 31
4.4.1 MQT Sebaran Mahasiswa 32
4.4.2 MQT Sebaran Mahasiswa berdasarkan Sekolah Asal 33
4.4.3 MQT Sebaran Mahasiswa berdasarkan Daerah Asal 35
4.5 Perancangan Staging Table 36
4.5.1 Staging Table Sebaran Mahasiswa 37
4.5.2 Staging Table Sebaran Mahasiswa berdasarkan Sekolah Asal 37
4.5.3 Staging Table Sebaran Mahasiswa berdasarkan Daerah Asal 38
BAB 5 IMPLEMENTASI 40
5.1 Implementasi Arsitektur 40
5.1.1 Lingkungan Perangkat Keras 40
5.1.2 Lingkungan Perangkat Lunak 40
5.1.3 Virtualisasi Server 40
5.2 Implementasi Basis Data 40
5.3 Implementasi Data Penelitian 41
5.3.1 Data Asli 41

5.3.2 Data Dummy 43
5.4 Implementasi Materialized Query Table 44
5.4.1 MQT Sebaran Mahasiswa 44
5.4.2 MQT Sebaran Mahasiswa berdasarkan Sekolah Asal 45
5.4.3 MQT Sebaran Mahasiswa berdasarkan Daerah Asal 47
5.5 Implementasi Staging Table 48
5.5.1 Staging Table Sebaran Mahasiswa 48
5.5.2 Staging Table Sebaran Mahasiswa berdasarkan Sekolah Asal 49
5.5.3 Staging Table Sebaran Mahasiswa berdasarkan Daerah Asal 50
BAB 6 PENGUJIAN DAN ANALISIS 52
6.1 Pengujian Performa 52
6.1.1 MQT Sebaran Mahasiswa 52
6.1.2 MQT Sebaran Mahasiswa berdasarkan Sekolah Asal 57
6.1.3 MQT Sebaran Mahasiswa berdasarkan Daerah Asal 62
BAB 7 PENUTUP 67
7.1 Kesimpulan 67
7.2 Saran 68
DAFTAR PUSTAKA 69
LAMPIRAN A DATA DEFINITION LANGUAGE (DDL) 70
LAMPIRAN B HASIL CEK PLAGIASI TURNITIN 74

DAFTAR TABEL

Tabel 4.1 Tabel-Tabel dalam Basis Data 30

Tabel 6.1 Skenario Pengujian MQT Sebaran Mahasiswa..... 52

Tabel 6.2 Hasil Perhitungan Waktu Eksekusi MQT Sebaran Mahasiswa..... 53

Tabel 6.3 Hasil Perhitungan Sumber Daya MQT Sebaran Mahasiswa 54

Tabel 6.4 Skenario Pengujian MQT Sebaran Mahasiswa berdasarkan Sekolah Asal 57

Tabel 6.5 Hasil Perhitungan Waktu Eksekusi MQT Sebaran Mahasiswa berdasarkan Sekolah Asal..... 57

Tabel 6.6 Hasil Perhitungan Sumber Daya MQT Sebaran Mahasiswa berdasarkan Sekolah Asal..... 58

Tabel 6.7 Skenario Pengujian MQT Sebaran Mahasiswa berdasarkan Daerah Asal 62

Tabel 6.8 Hasil Perhitungan Waktu Eksekusi MQT Sebaran Mahasiswa berdasarkan Daerah Asal..... 62

Tabel 6.9 Hasil Perhitungan Sumber Daya MQT Sebaran Mahasiswa berdasarkan Daerah Asal..... 63

DAFTAR GAMBAR

Gambar 2.1 Model Konseptual dari Database System 8
Gambar 2.2 Contoh Relasi..... 10
Gambar 2.3 Ilustrasi Pembuatan Tabel Asli (Sarraco, et al., 2012) 12
Gambar 2.4 Ilustrasi Pembuatan Tabel History (Sarraco, et al., 2012) 13
Gambar 2.5 Ilustrasi Tabel Versioning (Sarraco, et al., 2012)..... 13
Gambar 2.6 Ilustrasi Memasukkan Data kedalam Tabel dengan System Time (Sarraco, et al., 2012)..... 13
Gambar 2.7 Tabel Asli Setelah Insert Data (Sarraco, et al., 2012) 13
Gambar 2.8 Tabel History Setelah Insert Data (Sarraco, et al., 2012)..... 13
Gambar 2.9 Ilustrasi Mengubah Data kedalam Tabel dengan System Time (Sarraco, et al., 2012)..... 14
Gambar 2.10 Tabel Asli Setelah Update Data (Sarraco, et al., 2012)..... 14
Gambar 2.11 Tabel History Setelah Update Data (Sarraco, et al., 2012) 14
Gambar 2.12 Ilustrasi Menghapus Data kedalam Tabel dengan System Time (Sarraco, et al., 2012)..... 14
Gambar 2.13 Tabel Asli Setelah Delete Data (Sarraco, et al., 2012) 14
Gambar 2.14 Tabel History Setelah Delete Data (Sarraco, et al., 2012) 14
Gambar 2.15 Mengambil Data Saat Ini dari Tabel dengan System Time (Sarraco, et al., 2012)..... 15
Gambar 2.16 Mendapatkan Data pada Waktu Tertentu (Sarraco, et al., 2012). 15
Gambar 2.17 Mendapatkan Data pada Rentang Waktu Tertentu (Sarraco, et al., 2012) 15
Gambar 2.18 Ilustrasi Konseptual MQT 16
Gambar 2.19 Ilustrasi Penggunaan MQT Pada Aplikasi..... 17
Gambar 2.20 Ilustrasi inner join 19
Gambar 2.21 Perbedaan dari tipe outer join 19
Gambar 2.22 Definisi sintak untuk pemutakhiran MQT 20
Gambar 2.23 Definisi sintak untuk pembuatan staging table..... 22
Gambar 2.24 Transparent Query Rewrite (Lane & Potineni, 2014) 24
Gambar 3.1 Alur Metodologi Penelitian 25
Gambar 4.1 Rancangan Arsitektur MQT 29

Gambar 4.2 Diagram Rancangan Basis Data 30
Gambar 4.3 Diagram Rancangan MQT Sebaran Mahasiswa 32
Gambar 4.4 Aljabar Relasional MQT Sebaran Mahasiswa..... 33
Gambar 4.5 Diagram Rancangan MQT Sebaran Mahasiswa berdasarkan Sekolah Asal..... 34
Gambar 4.6 Aljabar Relasional MQT Sebaran Mahasiswa berdasarkan Sekolah Asal 34
Gambar 4.7 Diagram Rancangan MQT Sebaran Mahasiswa berdasarkan Daerah Asal..... 35
Gambar 4.8 Aljabar Relasional MQT Sebaran Mahasiswa berdasarkan Daerah Asal 36
Gambar 4.9 Diagram Rancangan Staging Table Sebaran Mahasiswa 37
Gambar 4.10 Diagram Rancangan Staging Table Sebaran Mahasiswa berdasarkan Sekolah Asal..... 38
Gambar 4.11 Diagram Rancangan Staging Table Sebaran Mahasiswa berdasarkan Daerah Asal..... 39
Gambar 5.1 Upload Data ke Server 41
Gambar 5.2 Memasukkan Data dengan Import Utility 42
Gambar 5.3 Jendela Import Utility 42
Gambar 5.4 Menampilkan Data Pada Tabel 43
Gambar 5.5 Data Hasil Proses Import..... 43
Gambar 5.6 Data Dummy Mahasiswa 44
Gambar 5.7 Implementasi Query MQT Sebaran Mahasiswa..... 45
Gambar 5.8 Implementasi Query MQT Sebaran Mahasiswa berdasarkan Sekolah Asal..... 46
Gambar 5.9 Implementasi Query MQT Sebaran Mahasiswa berdasarkan Daerah Asal..... 47
Gambar 5.10 Implementasi Query Staging Table Sebaran Mahasiswa..... 48
Gambar 5.11 Staging Table Sebaran Mahasiswa 49
Gambar 5.12 Implementasi Query Staging Table Sebaran Mahasiswa berdasarkan Sekolah Asal..... 49
Gambar 5.13 Staging Table Sebaran Mahasiswa berdasarkan Sekolah Asal..... 50
Gambar 5.14 Implementasi Query Staging Table Sebaran Mahasiswa berdasarkan Daerah Asal..... 50
Gambar 5.15 Staging Table Sebaran Mahasiswa berdasarkan Daerah Asal 51

Gambar 6.1 Grafik Perbandingan Eksekusi MQT Sebaran Mahasiswa 53

Gambar 6.2 Diagram Access Plan MQT Sebaran Mahasiswa dengan Full Refresh 55

Gambar 6.3 Diagram Access Plan MQT Sebaran Mahasiswa dengan Incremental Refresh 56

Gambar 6.4 Grafik Perbandingan Eksekusi MQT Sebaran Mahasiswa berdasarkan Sekolah Asal..... 58

Gambar 6.5 Diagram Access Plan MQT Sebaran Mahasiswa berdasarkan Sekolah Asal dengan Full Refresh 60

Gambar 6.6 Diagram Access Plan MQT Sebaran Mahasiswa berdasarkan Sekolah Asal dengan Incremental Refresh..... 61

Gambar 6.7 Grafik Perbandingan Eksekusi MQT Sebaran Mahasiswa berdasarkan Daerah Asal..... 63

Gambar 6.8 Diagram Access Plan MQT Sebaran Mahasiswa berdasarkan Daerah Asal dengan Full Refresh 65

Gambar 6.9 Diagram Access Plan MQT Sebaran Mahasiswa berdasarkan Daerah Asal dengan Incremental Refresh..... 66

DAFTAR LAMPIRAN

LAMPIRAN A DATA DEFINITION LANGUAGE (DDL).....70
LAMPIRAN B HASIL CEK PLAGIASI TURNITIN.....74

BAB 1 PENDAHULUAN

1.1 Latar belakang

Materialized Query Table (MQT) atau biasa disebut juga *materialized view* (MV) merupakan sebuah tabel yang berisikan data dari hasil pemrosesan suatu *query*. Pada dasarnya, *Data Definition Language* (DDL), atau *SQL code*, yang digunakan untuk membangun sebuah MQT mengandung *query-statement* sebagai intinya (Broughton, 2005). MQT berbentuk *data-caching* yang berguna untuk meningkatkan performa dari *database management system* dengan cara menyimpan hasil dari *query* yang sering digunakan (Gupta, et al., 2001). Data didalam sebuah MQT dihasilkan dengan proses eksekusi *query* pada suatu interval tertentu (periodik) atau pada suatu waktu yang spesifik dan seringkali berupa hasil agregasi atau perhitungan dari satu atau lebih tabel induk.

Keuntungan menggunakan MQT adalah untuk mengaksesnya hanya perlu membaca blok lokasi dimana *materialized query table* disimpan tanpa perlu melakukan komputasi ulang pada setiap pengaksesan (Paraboschi, et al., 2003). Hal ini tentu dapat meningkatkan performa sistem dengan mengurangi biaya *query* sehingga data dapat diakses dengan lebih cepat. Contohnya, ketika seorang manajer ingin mengetahui total penjualan selama jangka waktu tertentu pada masing-masing cabang. Saat data dikalkulasi melalui sebuah eksekusi *query*, pemrosesan dapat memakan waktu dari hitungan detik hingga jam setiap kali *statement query* diproses. MQT berperan untuk menyimpan data yang dihasilkan sehingga dapat digunakan berulang-ulang tanpa perlu meminta *database* untuk melakukan perhitungan ulang yang membutuhkan waktu lama dan memakan *resource* (Broughton, 2005).

Bentuk yang sederhana seperti tabel yang terbentuk dari hasil *query-statement* membuat MQT mudah untuk diimplementasikan. Terutama dengan dukungan yang diberikan oleh DBMS membuat proses yang terjadi dibalik perhitungan data dari tabel induk menuju ke MQT semakin mudah dan efisien. Pengguna dapat memperoleh data yang kompleks dengan biaya *query* yang lebih murah. Namun disamping keuntungan yang dimiliki oleh MQT juga terdapat kelemahan. Ketika data pada tabel induk mengalami perubahan, data yang ada didalam MQT menjadi usang. Hal itu dapat mengakibatkan data didalam MQT menjadi kurang relevan untuk digunakan sebagai bahan laporan analisis. Bahkan jika dibiarkan hal tersebut bisa berpotensi dampak buruk bagi pengguna, contohnya salah pengambilan keputusan oleh pihak majerial yang dapat mempengaruhi kelangsungan bisnis suatu instansi/organisasi. Oleh karena itu, dibutuhkan sebuah strategi yang efektif untuk menjaga agar data didalam MQT tetap terkini (*up-to-date*).

Secara garis besar, terdapat dua mekanisme untuk melakukan *refresh* MQT. Pertama, *Full Refresh*, yaitu menghapus seluruh data dalam MQT kemudian melakukan komputasi ulang seluruh data pada tabel induk untuk dimasukkan

kembali kedalam MQT. Kedua, *Incremental Refresh*, yaitu mekanisme *refresh* MQT dengan melakukan komputasi hanya kepada data-data yang berubah (delta) pada tabel induk (Lehner, et al., 2010). Namun seiring dengan pesatnya perkembangan data transaksional, mekanisme *incremental refresh* lebih sering dijadikan pilihan utama. Hal ini didasari oleh faktor banyaknya waktu yang dibutuhkan selama proses eksekusi *query* dan juga jumlah *resource* yang diperlukan saat proses *refresh* MQT berlangsung.

Berbeda dengan MQT dengan mekanisme *full refresh* yang mengkalkulasi ulang seluruh data pada tabel induk, MQT dengan *incremental refresh* hanya mempertimbangkan sebagian data didalam tabel induk yang mengalami perubahan. Hal ini dapat dilakukan dengan adanya sebuah *staging table*, yaitu sebuah table yang berfungsi untuk menyimpan seluruh perubahan data (delta) yang terjadi pada setiap table induk yang terhubung dengan MQT. *Staging table* memungkinkan MQT untuk melakukan pemutakhiran data dengan mengkalkulasi data-data yang mengalami perubahan tanpa harus mengakses seluruh data yang terdapat dalam table induk. Hal tersebut menyebabkan jumlah data yang harus diproses menjadi berkurang sehingga *query* dapat dilakukan dengan lebih efisien dan murah.

Bindu Sharma dan Mahesh Singh (2014) pada penelitiannya yang berjudul "*Performance Tuning in Database Management System based on Analysis of Combination of Time and Cost Parameter through Neural Network Learning*" menjelaskan bahwa meningkatkan performa *database management system* berarti meningkatkan performa dari *database* itu sendiri, salah satunya adalah dengan meminimalkan waktu respon dengan biaya sumber daya yang optimal. Pada dasarnya proses pemutakhiran MQT dengan mekanisme *incremental refresh* yang memproses hanya data-data yang berubah dinilai mampu untuk meminimalisir waktu respon dan juga mengoptimalkan penggunaan sumber daya dari *database*. Oleh sebab itu pada penelitian ini penulis ingin berfokus kepada pembahasan mengenai analisis serta implementasi *incremental refresh* untuk kemudian dibandingkan dengan mekanisme *full refresh* dalam proses pemutakhiran data didalam MQT.

Berdasarkan latar belakang yang telah dikemukakan maka penulis mengusulkan judul penelitian pada skripsi ini yaitu "*Incremental Refresh Materialized Query Table* (MQT) Memanfaatkan Staging Table Untuk Optimasi Query Execution Time Dan Resources Yang Digunakan".

1.2 Rumusan masalah

Berdasarkan latar belakang yang dikemukakan di atas maka rumusan permasalahan pada penelitian ini antara lain:

1. Bagaimana rancangan dan implementasi *full refresh* dan *incremental refresh* memanfaatkan *staging table* pada *materialized query table*?
2. Bagaimana pengaruh implementasi *incremental refresh* memanfaatkan *staging table* dibandingkan dengan *full refresh* terhadap peningkatan performa sistem dilihat dari segi waktu eksekusi *query*?
3. Bagaimana pengaruh implementasi *incremental refresh* menggunakan *staging table* dibandingkan dengan *full refresh* terhadap penggunaan *resources* dilihat dari CPU *utilization* dan *I/O process*?

1.3 Tujuan

Adapun tujuan dari penelitian ini antara lain:

1. Memodelkan dan mengimplementasikan metode *incremental refresh* memanfaatkan *staging table* serta membandingkannya dengan metode *full refresh* pada *materialized query table.*
2. Mengetahui pengaruh implementasi *incremental refresh* memanfaatkan *staging table* dibandingkan dengan *full refresh* terhadap peningkatan performa sistem dilihat dari segi waktu eksekusi query.
3. Mengetahui pengaruh implementasi *incremental refresh* memanfaatkan *staging table* dibandingkan dengan *full refresh* terhadap penggunaan *resources* dilihat dari CPU *utilization* dan *I/O process*.

1.4 Manfaat

Manfaat penelitian yang diharapkan dicapai dalam penelitian ini adalah sebagai berikut:

1. Bagi penulis, meningkatkan wawasan mengenai metode pemutakhiran data pada *materialized query table* serta pengaruh dari implementasi *incremental refresh* dibandingkan dengan *full refresh*.
2. Bagi Ilmu Pengetahuan, penelitian ini diharapkan dapat menjadi referensi bagi pengembangan ilmu terkait.
3. Bagi Masyarakat Umum, penelitian ini diharapkan dapat menjadi sumber wawasan mengenai pentingnya peranan *materialized query table* dan cara untuk melakukan pemutakhiran data yang ada didalamnya.

1.5 Batasan masalah

Berdasarkan latar belakang dan rumusan masalah yang telah diuraikan di atas, terdapat beberapa batasan masalah pada penelitian ini diantaranya:

1. Penelitian dilakukan menggunakan sistem manajemen basis data IBM yaitu IBM DB2.
2. Pengujian yang dilakukan adalah pengujian performa dari segi *query execution time* dan *resource* yang digunakan.
3. Penelitian berfokus pada pengaruh implementasi *incremental refresh* dibandingkan dengan *full refresh* pada MQT.
4. Penelitian tidak membahas seputar perubahan (*alter*) dari objek *database* lain diluar *materialized query table*.
5. Penelitian tidak membahas seputar kegagalan yang mungkin terjadi pada saat proses pemutakhiran data MQT yang disebabkan oleh hal teknis maupun non teknis.
6. Berdasarkan rekomendasi oleh dosen pembimbing maka data uji yang digunakan pada penelitian ini adalah data mahasiswa Universitas Brawijaya Malang yang diperoleh dari penelitian yang telah dilakukan sebelumnya.

1.6 Sistematika pembahasan

Sistematika penulisan skripsi ini adalah antara lain sebagai berikut:

BAB I: Pendahuluan
Bab ini berisi tentang latar belakang masalah, rumusan masalah, hipotesis, tujuan, manfaat dan batasan masalah dari penelitian serta sistematika penulisan skripsi.

BAB II : Kajian Pustaka dan Dasar Teori
Bab ini berisi tinjauan pustaka dari beberapa penelitian berkaitan sebelumnya serta landasan teori dilakukannya penelitian mengenai *incremental refresh materialized query table* dan lainnya.

BAB III : Metodologi
Bab ini berisi penjelasan dan metodologi yang akan digunakan pada penelitian ini meliputi: studi literatur, penggunaan dasar teori, analisis perancangan sistem, *tools* yang digunakan dalam penelitian serta metode pengujian yang digunakan pada penelitian ini.

BAB IV : Perancangan
Bab ini berisi penjelasan analisis yang telah dilakukan dan melakukan perancangan sistem.

BAB IV : Implementasi
Bab ini berisi implementasi perancangan sistem yang telah dilakukan sesuai metodologi yang ditetapkan.

BAB V : Pengujian

Bab ini berisi hasil pengujian efisiensi dan efektivitas implementasi *incremental refresh* dibandingkan dengan *full refresh* pada *materialized query table* serta pengaruhnya pada waktu eksekusi *query*.

BAB VI : Penutup

Dalam bab ini berisi tentang kesimpulan dan saran dari sistem yang berhubungan dengan permasalahan yang telah dibahas serta tindakan yang harus di ambil atas hasil penelitian.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Kajian pustaka pada penelitian ini membahas beberapa penelitian terdahulu terkait dengan mekanisme *refresh materialized query table* dan implementasinya untuk optimasi performa basis data.

Nica, Anisoara (2011) pada penelitiannya yang berjudul "*Incremental maintenance of materialized views with outerjoins*" menjelaskan bahwa saat ini pemrosesan data melalui pendekatan *outer-join* sudah semakin banyak digunakan pada *emerged system* seperti *Object-Relational Mapping* (ORM), *schema integration and exchange system*, dan *probabilistic database*. Penggunaan MQT dengan pendekatan *outer-joins* telah diperbolehkan pada banyak DBMS. Namun tanpa dukungan tentang mekanisme *incremental maintenance*. Untuk mengatasi masalah tersebut Anisoara mencoba mengusulkan algoritme yang dapat digunakan pada *Sybase SQL Anywhere RDBMS*. Terdapat beberapa kesimpulan yang dihasilkan oleh penelitiannya. Pertama, algoritme yang diusulkan dapat membuat satu *update statement* yang bekerja pada setiap MQT. Hal tersebut memungkinkan proses *incremental refresh* pada MQT dapat dilakukan secara lebih optimal demi tercapainya performa yang lebih baik. Kedua, algoritme tersebut memberikan dukungan kedalam sebuah *extended class of materialized views* terkait beberapa batasan yang terjadi saat proses pendefinisian MQT. Ketiga, karena *statement update* yang dihasilkan hanya satu, makan tidak diperlukan sebuah *temporary table* ketika proses *refresh* dilakukan.

Gupta dan Mumick (2006) pada penelitiannya yang berjudul "*Incremental Maintenance of Aggregate and Outerjoin Expressions*" menyebutkan bahwa dalam beberapa tahun terkahir banyak algoritme untuk *incremental maintenance* yang diusulkan. Namun belum ada satupun algoritme yang mampu menyelesaikan kasus dari *relational expression* menggunakan *aggregate* dan *outerjoin operators* secara efisien. Oleh karena itu penelitiannya mengusulkan sebuah teknik *change-table* untuk melakukan *incremental maintenance* pada *general view expressions involving relational and aggregate operator* yang diklaim mampu berjalan lebih baik dibandingkan dengan algoritme-algoritme yang telah diusulkan sebelumnya. *Framework* yang dihasilkan dari penelitian tersebut mudah berkembang dan secara efisien menangani *maintenance* dari *views* yang mengandung *outerjoin operator.* Penelitiannya juga membuktikan bahwa teknik *change-table* merupakan mekanisme optimal dari proses *incremental maintenance* pada sebuah *expression* didasarkan pada beberapa asumsi tertentu.

Larson dan Zhou (2007) pada penelitiannya yang berjudul "*Efficient Maintenance of Materialized Outer-Join Views*" menyebutkan bahwa mekanisme *outerjoin* dapat memberikan dampak yang signifikan terhadap kecepatan dari proses *query* namun hampir seluruh *database system* tidak mengijinkan *outerjoin* untuk dilakukan didalam *materialized views*. Hal tersebut disebabkan karena

outer-joins tidak dapat di *maintenance* secara efisien ketika data pada tabel induk/rujukan mengalami perubahan. Penelitiannya menunjukan bagaimana ketika tabel induk mengalami perubahan *views* yang dibangun dari kumpulan proses *selection, projection, inner* dan *outer joins* dapat di *maintenance* secara efisien. Ekploitasi dari *foreign-key constraint* digunakan untuk mengurangi beban pada saat proses *maintenance* berlangsung. Hasil dari eksperimen menunjukkan bahwa *maintenance* sebuah *view* dengan pendekatan *outer-joins* tidak selalu memerlukan biaya operasional yang lebih tinggi dibandingakan *view* dengan pendekatan *inner-joins*.

Nugroho (2015) pada penelitiannya yang berjudul "Implementasi *Materialized Query Table* Untuk Pembangunan *Data Mart* (Studi Kasus: Universitas Brawijaya Bagian Akademik)" menyebutkan bahwa peran utama data mart adalah sebagai penyedia data bahan analisis untuk mendukung proses pengambilan keputusan. Yang pada pelaksanaannya akan melibatkan serangkaian query-query kompleks yang dilakukan berulang-ulang dan diakses oleh banyak pengguna. Di sinilah materialized query table akan berperan untuk membantu melakukan efisiensi dalam proses pengambilan dan pengoperasian data. Penelitiannya melakukan pembandingan performa dan ketahanan query yang melalui implementasi materialized query table dan query yang langsung mengakses ke data mart. Sebagian data pada penelitiannya diambil dan digunakan pada penelitian ini sebagai data asli.

Berdasarkan penelitian-penelitian yang dijabarkan di atas maka penulis dapat menyimpulkan bahwa dengan penggunaan *materialized query table* sudah semakin banyak digunakan. Seperti yang dijelaskan sebelumnya bahwa sebuah MQT dapat memberikan dampak signifikan terhadap kecepatan dan biaya yang diperlukan untuk pemrosesan *query*. Namun sayangnya dalam kasus MQT dengan *outer-joins* masih belum memperoleh dukungan langsung dari DBMS sehingga banyak batasan-batasan terutama masalah bagaimana menjaga agar data didalam MQT tetap selalu terkini (*up-to-date*). Jika dapat dibangun sebuah mekanisme yang dapat mengatasi batasan-batasan tersebut maka diharapkan proses analisis akan dapat dilakukan dengan lebih cepat dan efisien.

2.2 Database System

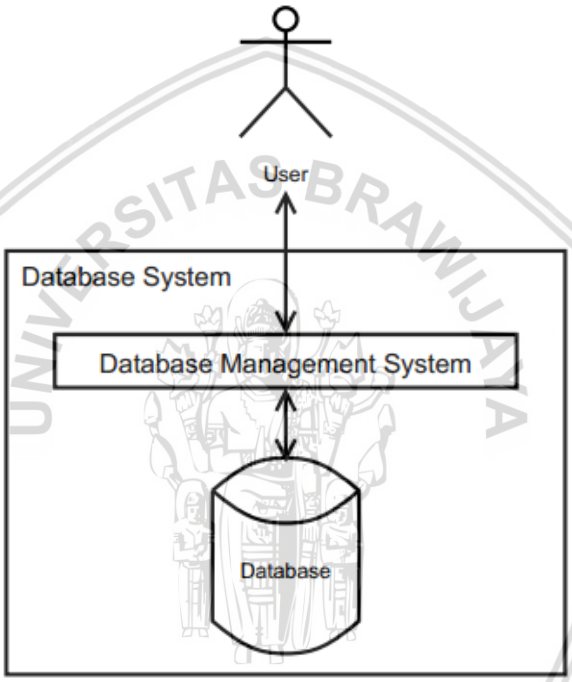
Database (DB) merupakan sebuah tempat penyimpanan data, di desain secara efisien untuk mendukung penyimpanan, pengambilan serta pemutakhiran data (Sharma, et al., 2010). Seluruh akses ke *database* akan melalui *Database Management System* (DBMS). Seperti pada Gambar 2.2.1 sebuah *database* yang dikelola oleh DBMS dikenal dengan istilah *database system* (Elmasri & Navathe, 2004).

Akses kedalam *database* dijalankan dengan eksekusi sebuah program transaksi, proses ini ditandai dengan *start* dan *end* yang telah didefinisikan, dimulai dengan operasi *start* dan diakhiri dengan *commit* atau *abort*. Proses *commit* memastikan bahwa semua operasi yang ada dalam lingkup transaksi telah

diproses dan disimpan dengan aman, sedangkan *abort* mengembalikan perubahan yang dilakukan oleh transaksi (Elmasri & Navathe, 2004).

Pada penggunaan secara umum, sebuah transaksi memiliki 4 komponen utama yang disebut dengan "ACID" (Haerder & Reuter, 1983):

- ***Atomicity*** – sebuah transaksi harus dieksekusi secara sukses, atau tidak dieksekusi sama sekali. Oleh karena itu DBMS harus mampu mengembalikan seluruh perubahan yang dilakukan oleh transaksi yang dibatalkan.



Gambar 2.1 Model Konseptual dari Database System

- ***Consistency*** – sebuah transaksi harus merubah *database* dari status konsisten satu ke status konsisten yang lain.
- ***Isolation*** – setiap transaksi berjalan secara mandiri dan tidak mengganggu antara satu dengan yang lain.
- ***Durability*** – hasil dari sebuah transaksi adalah permanen saat terjadi proses *commit*.

Saat ini sudah berbagai macam bentuk *database* dibangun untuk memenuhi kebutuhan industri. Sebuah *database* dapat secara spesifik digunakan untuk menyimpan *binary files,* dokumen, gambar, video, *relational data, multidimentional data, transactional data, analitycal data* ataupun bentuk-bentuk representasi data yang lain. Saat ini *database* telah menjadi aplikasi komputer yang paling penting dan paling luas penggunaannya. Sebaik apapun aplikasi antar mukanya jika tidak didukung oleh *Database Manajemen System* (DBMS) yang

handal tidak akan dapat bekerja secara maksimal. Ketergantungan yang sangat besar dari aplikasi terhadap sistem penyimpanan data menuntut DBMS untuk dapat menyediakan performa tinggi, ketersediaan data yang baik, kemudahan pengelolaan serta kehandalan dalam pemrosesan data.

Pada kenyataannya, performa dari sebuah aplikasi telah menjadi kebutuhan yang mutlak. Banyak tantangan dalam proses mencapai sebuah kinerja yang baik dalam hal ini adalah kemampuan pengelolaan data oleh *database*. DBMS didukung oleh fitur-fitur yang dapat membantu untuk memenuhi tuntutan tersebut. Salah satunya yang saat ini semakin sering digunakan adalah pemanfaatan *Materialized Query Table* (MQT). MQT merupakan sebuah mekanisme yang digunakan untuk meningkatkan performa dari Online Transactional Processing (OLTP) ataupun Online Analytical Processing (OLAP) (Satish, et al., 2007).

2.3 Database Management System

Secara umum *database* merupakan tempat penyimpanan data, sedangkan *Database Management System* (DBMS) adalah sebuah perangkat lunak yang berfungsi untuk mengontrol, mengorganisir, menyimpan, memanajemen, mendapatkan, dan menjaga data didalam *database* (Sharma, et al., 2010).

Sebuah DBMS memiliki peranan yang sangat besar dalam proses manajemen data didalam *database*. Pada dasarnya data dalam *database* akan selalu berubah-ubah seiring dengan permintaan dari pengguna. Operasi *insert, update, delete* akan dijalankan oleh aplikasi untuk memanipulasi data berdasarkan kebutuhan pengguna. Ketika terdapat banyak pengguna yang melakukan berbagai operasi manipulasi data pada suatu waktu, harus dipastikan bahwa masing-masing pengguna tidak mengganggu operasi yang dilakukan oleh pengguna lain yang dapat menyebabkan data menjadi tidak konsisten. Terlebih juga diperlukan mekanisme *backup and restore* untuk memastikan bahwa data aman bahkan ketika terjadi kegagalan sistem. Kebutuhan-kebutuhan tersebut tidak dapat dilakukan secara mandiri oleh *database*. Oleh karena itu DBMS dirancang untuk dapat memenuhi kebutuhan dalam pengelolaan data didalam *database* (Sharma, et al., 2010).

Saat ini telah banyak perusahaan yang membuat DBMS untuk memenuhi kebutuhan pengelolaan data seperti IBM DB2, Oracle, MySQL, PostgreSQL, MariaDB, MongoDB, SQL Server. Masing-masing memiliki peranannya sendiri dalam memanajemen *user* dan data didalam *database*. Pada penelitian ini DBMS yang digunakan adalah IBM DB2 10.5 *Enterprise Edition*.

2.4 Aljabar Relasional

Untuk mempermudah proses perancangan *materialized query table* maka akan digunakan bantuan aljabar relasional. Aljabar relasional merupakan sekumpulan operasi dasar untuk memanipulasi data pada model relasional

dengan tujuan untuk mendapatkan informasi yang diinginkan (Silberschatz, et al., 2009). Istilah **Relasi**, dalam bahasan ini dipergunakan untuk penamaan tabel beserta datanya baik yang murni maupun yang sudah dilakukan modifikasi dengan operasi-operasi aljabar relasional.

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	58
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotltaw	28

Gambar 2.2 Contoh Relasi

Secara umum aljabar relasional dibagi kedalam beberapa operasi dasar yaitu *select*, *project*, *union*, *set difference*, *cartessian product* dan *rename*. Serta operasi tambahan yaitu *set intersection*, *natural join* dan *assignment*. Berikut ini adalah penjelasan dari operasi-operasi dasar tersebut:

- Operasi *SELECT*

  Digunakan untuk memilih baris tertentu dari sebuah himpunan baris data (record) yang memenuhi kondisi dan membuang baris yang lain. Notasi yang digunakan untuk melambangkan operasi ini adalah sigma (σ).

- Operasi *PROJECT*

  Digunakan untuk memilih attribut (kolom) tertentu dari relasi / sub relasi dan membuang yang lain. Notasi yang digunakan untuk melambangkan operasi ini adalah *Pi* (∏).

- Operasi *UNION*

  Digunakan untuk menggabungkan dua relasi atau *result set* yang memiliki kriteria kolom yang sama. Notasi yang digunakan untuk melambangkan operasi ini adalah karakter (U).

- Operasi *SET DIFFERENCE*

  Digunakan untuk melakukan pencarian tupel (baris) yang hanya berada pada satu relasi. Notasi yang digunakan untuk melambangkan operasi ini adalah karakter minus (-).

- Operasi *CARTESSIAN PRODUCT*

  Digunakan untuk mengombinasikan informasi dari dua relasi. Notasi yang digunakan untuk melambangkan operasi ini adalah karakter *cross* (x).

- Operasi *RENAME*

  Digunakan untuk memberi nama pada bagian relasi. Notasi yang digunakan untuk melambangkan operasi ini adalah *rho* kecil (ρ).

- Operasi *SET INTERSECTION*

  Digunakan untuk memilih data yang menjadi bagian dari dua relasi atau lebih. Notasi yang digunakan untuk melambangkan operasi ini adalah karakter ($\cap$).

- Operasi *NATURAL JOIN*

  Digunakan untuk mengombinasikan pilihan tertentu dengan *Cartessian Product* ke dalam satu operasi. Notasi yang digunakan untuk melambangkan operasi ini adalah simbol join ($\bowtie$).

- Operasi *ASSIGMENT*

  Digunakan untuk menampung hasil operasi pada variabel sementara. Notasi yang digunakan untuk melambangkan operasi ini adalah simbol ($\leftarrow$).

2.5 Temporal Data Management

Teknologi pengelolaan data sementara (*Temporal Data Mangement*) memungkinkan perusahaan untuk melacak dan melakukan seleksi (*query)* dari data lampau, data saat ini, dan masa depan dengan cara yang mudah dan efisien. Tujuannya adalah untuk mempermudah audit data dan merencakan strategi bisnis kedepannya, untuk menentukan dan memperbaiki *human-errors*, untuk memastikan integritas data dari waktu ke waktu, dan untuk menilai perubahan kondisi bisnis (Sarraco, et al., 2012). Teknologi ini telah di implementasikan pada DBMS IBM DB2 versi 10.

2.5.1 Keuntungan *Temporal Data Management*

Teknologi ini didukung secara langsung oleh DB2 sehingga pengguna tidak perlu melakukan proses *query* yang rumit untuk mendapatkan data baik data pada masa lalu maupun data saat ini. Hanya dengan menggunakan deklarasi pernyataan SQL sederhana, *administrator* dapat menginstruksikan DB2 untuk menjaga sejarah data berdasarkan tanggal perubahannya tanpa perlu menggunakan *stored*

procedure, *trigger*, ataupun aplikasi tertentu yang rumit. Hal ini membuat proses perekaman data dapat dilakukan secara cepat dan mudah (Sarraco, et al., 2012).

2.5.2 Manajemen Versi Data Menggunakan dengan *System Time*

Dukungan oleh DB2 memungkinkan kita untuk secara otomatis merekam dan mengatur bermacam versi data. Membuat tabel dengan sebuah *system time period* berarti kita telah menginstruksikan DB2 untuk secara otomatis merekam semua perubahan data didalam tabel dan menyimpan data lama pada sebuah *history table*, yaitu sebuah tabel yang memiliki struktus yang sama dengan tabel saat ini. *Temporal query* yang diarahkan langsung pada tabel saat ini akan menyebabkan DB2 mengakses *history table* ketika diperlukan. Hal ini memungkinkan kita untuk bekerja dengan data pada masa lampau secara mudah, tanpa perlu untuk menuliskan pernyataan *query* yang kompleks (Sarraco, et al., 2012).

2.5.2.1 Membuat tabel dengan memanfaatkan *system time*

Proses pembuatan tabel dengan *system time* secara garis besar sama seperti pembuatan tabel biasa, hanya perlu ditambakan beberapa perubahan agar tabel tersebut dapat menggunakan *history tabel*. Langkah-langkahnya adalah sebagai berikut:

1. Membuat tabel asli/induk untuk menyimpan data saat ini
   Mendefinisikan semua kolom yang dibutuhkan, kemudian menambahkan 3 kolom baru dengan tipe data TIMESTAMP(12) yang masing-masing berguna untuk menyimpan waktu saat data itu dibuat, waktu saat data itu mengalami perubahan serta waktu saat pertama kali data itu masuk kedalam tabel. Kemudian mendefinisikan 3 kolom tersebut sebagai GENERATED ALWAYS agar DB2 secara otomatis mengisikan *value* pada saat terjadi perubahan data seperti diilustrasikan pada gambar 2.3.

```
CREATE TABLE policy (
   id              INT primary key not null,
   vin             VARCHAR(10),
   annual_mileage INT,
   rental_car      CHAR(1),
   coverage_amt    INT,
   sys_start       TIMESTAMP(12) GENERATED ALWAYS AS ROW BEGIN NOT NULL,
   sys_end         TIMESTAMP(12) GENERATED ALWAYS AS ROW END NOT NULL,
   trans_start     TIMESTAMP(12) GENERATED ALWAYS
                                 AS TRANSACTION START ID IMPLICITLY HIDDEN,
 PERIOD SYSTEM_TIME (sys_start, sys_end)
);
```

Gambar 2.3 Ilustrasi Pembuatan Tabel Asli (Sarraco, et al., 2012)

2. Membuat tabel *history*
   Membuat tabel yang identik dengan tabel asli/induk, bisa dilakukan dengan menggunakan pernyataan CREATE TABEL ... LIKE ... seperti pada gambar 2.4.

```
CREATE TABLE policy_history LIKE policy;
```

Gambar 2.4 Ilustrasi Pembuatan Tabel History (Sarraco, et al., 2012)

3. Mengubah tabel asli untuk mengaktifkan *versioning* dan mengidentifikasi *history table* seperti diilustrasikan pada gambar 2.5.

```
ALTER TABLE policy ADD VERSIONING USE HISTORY TABLE policy_history;
```

Gambar 2.5 Ilustrasi Tabel Versioning (Sarraco, et al., 2012)

2.5.2.2 Memasukkan data kedalam tabel yang menggunakan *system time*

Cara memasukkan data kedalam tabel yang telah memanfaatkan *system time* sama dengan memasukkan data kedalam tabel biasa, hanya saja 3 kolom *system time* yang telah didefinisikan sebelumnya akan otomatis terisi, seperti pada gambar 2.6.

```
INSERT INTO policy(id, vin, annual_mileage, rental_car, coverage_amt)
    VALUES(1111, 'A1111',  10000, 'Y', 500000);

INSERT INTO policy(id, vin, annual_mileage, rental_car, coverage_amt)
    VALUES(1414,  'B7777',  14000, 'N', 750000);
```

Gambar 2.6 Ilustrasi Memasukkan Data kedalam Tabel dengan System Time (Sarraco, et al., 2012)

Sedangkan berikut ini merupakan kondisi tabel asli dan tabel *history* setelah data tersebut dimasukkan, seperti pada gambar 2.7 dan 2.8.

ID	VIN	annual_mileage	rental_car	coverage_amt	sys_start	sys_end
1111	A1111	10000	Y	500000	2010-11-15	9999-12-30
1414	B7777	14000	N	750000	2010-11-15	9999-12-30

Gambar 2.7 Tabel Asli Setelah Insert Data (Sarraco, et al., 2012)

ID	VIN	annual_mileage	rental_car	coverage_amt	sys_start	sys_end

Gambar 2.8 Tabel History Setelah Insert Data (Sarraco, et al., 2012)

Bisa diperhatikan bahwa belum ada data pada tabel *history* dikarenakan belum ada perubahan data apapun pada tabel asli karena operasi yang dilakukan adalah memasukkan data.

2.5.2.3 Mengubah data dari tabel yang menggunakan *system time*

Mengubah data pada tabel yang telah memanfaatkan *system time* juga tidak berbeda dengan mengubah data pada tabel biasa, seperti pada gambar 2.9.

```
UPDATE policy
SET coverage_amt = 750000
WHERE id = 1111;
```

Gambar 2.9 Ilustrasi Mengubah Data kedalam Tabel dengan System Time (Sarraco, et al., 2012)

Sedangkan berikut ini merupakan kondisi tabel asli dan tabel *history* setelah data tersebut dimasukkan, seperti pada gambar 2.10 dan 2.11.

ID	VIN	annual_mileage	rental_car	coverage_amt	sys_start	sys_end
1111	A1111	10000	Y	750000	2011-01-31	9999-12-30
1414	B7777	14000	N	750000	2010-11-15	9999-12-30

Gambar 2.10 Tabel Asli Setelah Update Data (Sarraco, et al., 2012)

ID	VIN	annual_mileage	rental_car	coverage_amt	sys_start	sys_end
1111	A1111	10000	Y	500000	2010-11-15	2011-01-31

Gambar 2.11 Tabel History Setelah Update Data (Sarraco, et al., 2012)

Bisa diperhatikan tabel *history* menyimpan data yang telah diubah sekaligus informasi berupa tanggal tentang kapan data tersebut dimulai dan berakhir yang secara otomatis dibuat oleh DBMS.

2.5.2.4 Menghapus data dari tabel yang menggunakan s*ystem time*

Menghapus data pada tabel yang telah memanfaatkan *system time* juga tidak berbeda dengan mengubah data pada tabel biasa, seperti pada gambar 2.12.

```
DELETE FROM policy WHERE id = 1414;
```

Gambar 2.12 Ilustrasi Menghapus Data kedalam Tabel dengan System Time (Sarraco, et al., 2012)

Sedangkan berikut ini merupakan kondisi tabel asli dan tabel *history* setelah data tersebut dimasukkan, seperti pada gambar 2.13 dan 2.14.

ID	VIN	annual_mileage	rental_car	coverage_amt	sys_start	sys_end
1111	A1111	5000	N	250000	2012-01-31	9999-12-30

Gambar 2.13 Tabel Asli Setelah Delete Data (Sarraco, et al., 2012)

ID	VIN	annual_mileage	rental_car	coverage_amt	sys_start	sys_end
1111	A1111	10000	Y	500000	2010-11-15	2011-01-31
1414	B7777	14000	N	750000	2010-11-15	2012-03-31

Gambar 2.14 Tabel History Setelah Delete Data (Sarraco, et al., 2012)

Bisa diperhatikan tabel *history* menyimpan data yang telah dihapus sekaligus informasi berupa tanggal tentang kapan data tersebut dimulai dan berakhir yang secara otomatis dibuat oleh DBMS.

2.5.2.5 Menghapus data dari tabel yang menggunakan s*ystem time*

Mengambil (query*)* data terkini dari tabel yang telah memanfaatkan *system time* juga tidak berbeda dengan mengubah data pada tabel biasa, seperti pada gambar 2.15.

```
SELECT coverage_amt FROM policy WHERE id = 1111;
```

Gambar 2.15 Mengambil Data Saat Ini dari Tabel dengan System Time (Sarraco, et al., 2012)

Namun kondisi tersebut akan sedikit berbedia ketika ingin menampilkan data dari masa lampau yang mengharuskan kita untuk mengakses data dari dalam tabel *history*. DB2 telah mendukung beberapa pernyataan *query* yang secara khusus didesain untuk mengambil data dari dalam tabel *history* sesuai dengan kebutuhan pengguna, adapun pernyataan tersebut adalah sebagai berikut:

- FOR SYSTEM_TIME AS OF ...
  Digunakan untuk mendapatkan data pada suatu waktu tertentu

```
SELECT coverage_amt
FROM policy FOR SYSTEM_TIME AS OF '2010-12-01'
WHERE id = 1111;
```

Gambar 2.16 Mendapatkan Data pada Waktu Tertentu (Sarraco, et al., 2012)

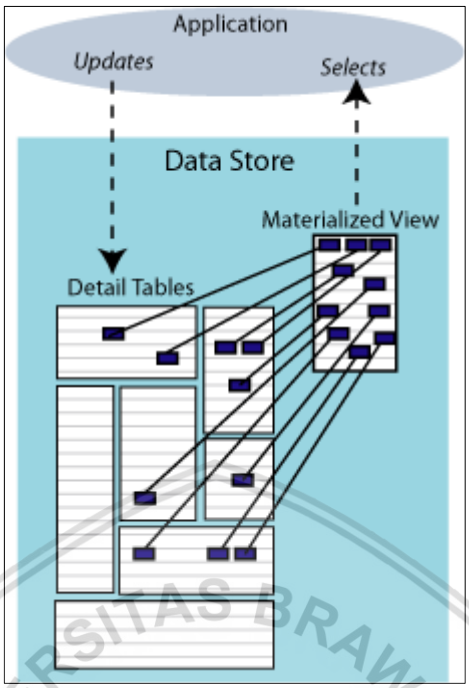
- FOR SYSTEM_TIME FROM ... TO ...
  Digunakan untuk mendapatkan data dari waktu tertentu sampai waktu tertentu, dimana *start-times* harus berada pada *range* data dalam tabel sedangkan *end-times* diperbolehkan untuk waktu kapanpun.

```
SELECT count(*)
FROM policy FOR SYSTEM_TIME FROM '2011-11-30'
                            TO '9999-12-30'
WHERE vin = 'A1111';
```

Gambar 2.17 Mendapatkan Data pada Rentang Waktu Tertentu (Sarraco, et al., 2012)

2.6 Materialized Query Table

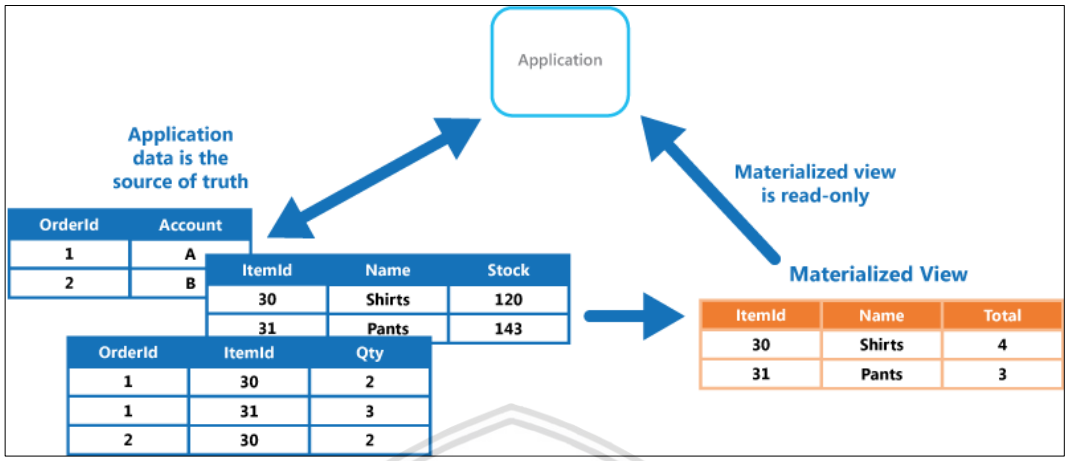
Materialized Query Table (MQT) adalah sebuah objek basis data berupa tabel yang berisikan data hasil pemrosesan suatu *query*. Pada dasarnya, *Data Definition Language* (DDL), atau *SQL code*, yang digunakan untuk membangun sebuah MQT mengandung *query-statement* sebagai intinya (Broughton, 2005). MQT berbentuk *data-caching* yang dapat digunakan untuk meningkatkan performa dari *database management system* dengan cara menyimpan hasil dari *query* yang sering digunakan (Gupta, et al., 2001). Data didalam sebuah MQT dihasilkan dengan proses eksekusi *query* pada suatu interval tertentu (periodik) atau pada suatu waktu yang spesifik dan seringkali berupa hasil agregasi atau kesimpulan dari satu atau lebih tabel induk.



Gambar 2.18 Ilustrasi Konseptual MQT

MQT sangat mirip dengan tabel pada umumnya, hanya saja dibentuk melalui hasil dari proses *query*. Banyak digunakan di lingkungan dimana performa adalah hal yang diutamakan. MQT memiliki peranan yang besar di lingkungan yang banyak melakukan pemrosesan data dalam jumlah sangat besar dan *query* yang hampir sama dilakukan secara berulang-ulang. Jika *query* tersebut di proses lebih awal (*pre-computed*) dan hasilnya disimpan kedalam *database* sebagai sebuah MQT, maka dengan memanfaatkannya dapat meningkatkan performa secara signifikan. Sebuah MQT digunakan untuk mengurangi beban yang disebabkan oleh penggabungan (*join*) dan agregasi data dengan biaya yang mahal. Tujuannya adalah untuk meningkatkan performa eksekusi *query*.

Bayangkan jika pada sebuah perusahaan *retail* yang memiliki persediaan barang sangat besar. Ketika manajer ingin mengetahui jumlah dari setiap barang yang terjual maka sistem harus melakukan perhitungan terhadap data dengan jumlah yang sangat banyak dan belum lagi jika ternyata data-data tersebut disimpan dalam tabel yang terpisah. Terlebih jika permintaan ini dilakukan secara berulang-ulang maka akan sangat membebani sistem. Disinilah MQT berperan untuk melakukan *pre-computed* dari data-data tersebut dan kemudian menyimpannya kedalam *database* untuk dimanfaatkan setelahnya. Seperti di ilustrasikan pada Gambar 2.19, aplikasi akan sangat dimudahkan dengan hanya perlu meminta data dari dalam MQT dibandingkan dengan harus menggabungkan tabel-tabel terkait kemudian mulai menghitungnya. Hal tersebut tentu akan berpengaruh pada beban sistem dilihat dari segi performa *query* dan *resource* yang diperlukan untuk memperoleh data. Selama data pada tabel induk tidak berubah maka data yang berada dalam MQT akan tetap valid.



Gambar 2.19 Ilustrasi Penggunaan MQT Pada Aplikasi

Keberadaan MQT sangat jelas pada DBMS, sehingga seorang *database administrator* dapat membuat atau menghapusnya dengan mudah tanpa harus mempengaruhi objek-objek yang lain.

2.6.1 Keuntungan MQT

MQT menawarkan sebuah cara untuk meningkatkan *response time* dari *query* yang kompleks, terlebih ketika proses *query* dijalankan pada beberapa operasi berikut:

- Agregasi data melalui satu atau lebih dimensi.
- Penggabungan dan agregasi data dari beberapa tabel sekaligus.
- Menjalankan perhitungan yang berulang-ulang.
- Menjalankan pemindaian yang memakan *resource.*
- Mengakses data dari tabel, atau bagian dari tabel, pada sebuah lingkungan *partitioned-database*.

Semakin besar tabel-tabel induk yang berhubungan dengan MQT, semakin besar juga potensi peningkatan *response time*, dikarenakan MQT tumbuh lebih lambat dibandingkan dengan tabel induknya (Sanders, 2009).

Berikut ini beberapa aplikasi yang memperoleh keuntungan dari dibuatnya sebuah MQT:

- *Decision Support System*
- *Data Marts*
- *Data Warehouses*
- *Online Analytical Processing* (OLAP)
- *Data Mining Workload*

2.6.2 Jenis-Jenis MQT

Berikut ini merupakan jenis-jenis MQT berdasarkan pengolahan data didalamnya (Melnyk, 2005), yaitu:

- MQT yang Dikelola Oleh Sistem

  Data yang terdapat didalam MQT jenis ini seluruhnya dikelola oleh sistem. Klausa MAINTENED BY SYSTEM digunakan pada saat MQT didefinisikan. Ketika mendefinisikan MQT, kita dapat menentukan mekanisme pemutakhiran data antar REFRESH IMMEDIATE atau REFRESH DEFERRED. Klausa REFRESH mengijinkan kita untuk menentukan bagaimana data didalam MQT akan dimutakhirkan. Klausa DEFERRED berarti bahwa data didalam tabel dapat dimutakhirkan kapan saja dengan menggunakan klausa REFRESH TABLE. Jenis ini tidak mengijinkan pengguna untuk melakukan operasi *insert, update, delete* secara langsung terhadap MQT. Namun, REFRESH IMMEDIATE akan secara otomatis mengaplikasikan perubahan yang terjadi pada tabel induk yang disebabkan oleh operasi *insert, update, delete* secara langsung kedalam MQT pada saat perubahan tersebut terjadi.

- MQT yang Dikelola Oleh Pengguna (User)

  Data yang terdapat pada MQT jenis ini seluruhnya dikelola oleh pengguna. Klausa MAINTENED BY USER digunakan untuk mendefinisikan MQT. Hanya klausa REFRESH DEFERRED yang dapat digunakan pada saat pendefinisian MQT. Jenis ini memungkinkan pengguna untuk melakukan operasi *insert, update, delete* secara langsung terhadap MQT. Secara sederhana, MQT ini berperilaku sama persis layaknya tabel biasa, kecuali dengan beberapa kelebihan yang dimiliki oleh MQT seperti penggunaan *query rewrite* yang akan dibahas selanjutnya.

Pada penelitian ini jenis MQT yang digunakan adalah *user-maintened* MQT atau MQT yang dikelola oleh pengguna. Hal ini karena pada proses *incremental maintenance* MQT akan dikelola secara penuh oleh pengguna.

2.6.3 Join Table Pada MQT

Join atau penggabungan merupakan mekanisme untuk menggabungkan data dari dua atau lebih tabel, *view,* atau sesama *materialized query table* melalui sebuah kondisi penggabungan. DBMS melakukan proses *join* ketika ada lebih dari satu tabel berada pada klausa `FROM` dari sebuah *query* (Oracle, 2015).

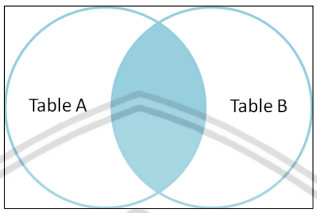
2.6.3.1 *Join Condition*

Pada kebanyakan kasus, *join-query* selalu memiliki setidaknya satu kondisi penggabungan, kondisi tersebut bisa terdapat pada klausa `FROM` atau `WHERE`. Kondisi ini membandingkan dua kolom dari masing-masing tabel. Pada saat melakukan *join*, DBMS mencocokkan data dari masing-masin tabel sesuai kondisi

yang diberikan. Pada saat kondisi terpenuhi maka data pada seluruh kolom yang diseleksi akan ditampilkan (Oracle, 2015).

2.6.3.2 *Inner Join*

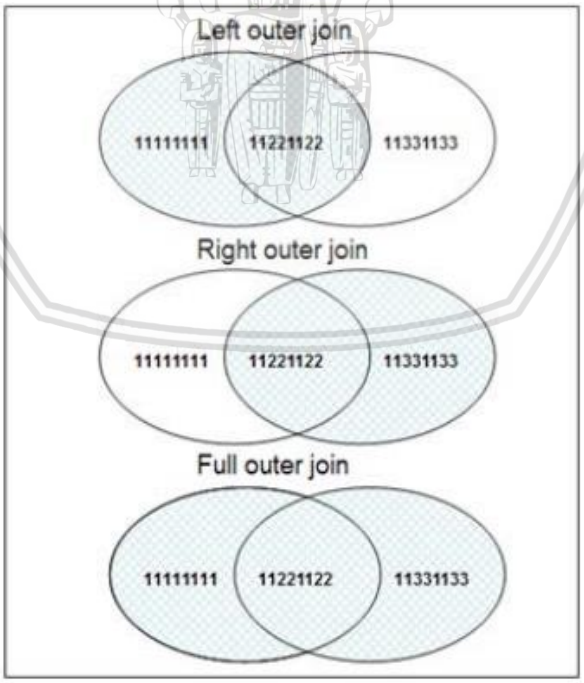
Penggabungan ini hanya akan menampilkan data-data yang memenuhi *join condition* (Oracle, 2015). Dapat dikatakan bahwa data yang ditampilkan hanyalah yang memiliki hubungan secara langsung dengan tabel lain yang masuk dalam penggabungan. Ilustrasi penggabungan tipe ini seperti dapat dilihat pada gambar 2.20.



Gambar 2.20 Ilustrasi inner join

2.6.3.3 *Outer Join*

Penggabungan ini merupakan bentuk spesial dari *join* yang digunakan pada *query*. Dalam kasus *outer join*, tabel pertama yang berada pada klausa FROM akan dianggap sebagai tabel KIRI, sedangkan tabel yang selanjutnya dianggap sebagai tabel KANAN (Sharma, et al., 2010).



Gambar 2.21 Perbedaan dari tipe outer join

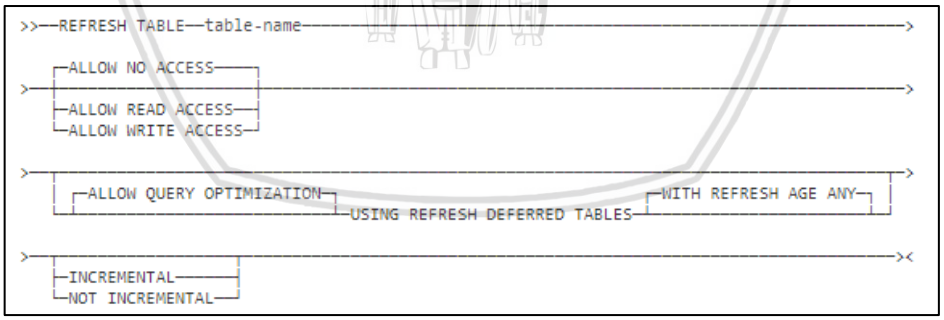
Berdasarkan ilustrasi pada gambar 2.21, berikut merupakan beberapa jenis dari *outer join* (Sharma, et al., 2010):

- Left Outer Join
  Tipe ini akan menampilkan data-data yang berhubungan langsung antara tabel kiri dan tabel kanan serta data pada tabel kiri walaupun tidak memiliki pasangan pada tabel kanan.
- Right Outer Join
  Tipe ini akan menampilkan data-data yang berhubungan langsung antara tabel kiri dan tabel kanan serta data pada tabel kanan walaupun tidak memiliki pasangan pada tabel kiri.
- Full Outer Join
  Tipe ini akan menampilkan data-data yang berhubungan langsung antara tabel kiri dan tabel kanan serta data pada tabel kiri dan kanan walaupun tidak memiliki pasangan pada tabel lainnya yang masuk dalam penggabungan.

Dari kedua ilustrasi pada Gambar 2.20 dan Gambar 2.21 dapat dilihat bahwa data yang ditampilan oleh *inner* dan *outer join* berbeda. Pada beberapa kondisi, tipe *outer join* memiliki keuntungan karena dapat menampilkan data yang tidak dapat dilakukan oleh tipe *inner join* sehingga informasi yang didapat menjadi lebih besar. Pada penelitian ini tipe penggabungan yang digunakan adalah *outer join*.

2.6.4 Mekanisme Pemutakhiran MQT

Database memelihara data didalam MQT dengan cara pemutakhiran (*refresh*) ketika terjadi perubahan pada tabel induk. Metode *refresh* dapat dilakukan secara *incremental* atau dengan *complete refresh*. Secara umum metode *incremental refresh* mempunyai performa yang lebih baik dibandingkan dengan *complete refresh* (Lane & Potineni, 2014).



Gambar 2.22 Definisi sintak untuk pemutakhiran MQT

Pada saat pilihan INCREMENTAL ataupun NOT INCREMENTAL tidak didefinisikan maka system akan secara otomatis mendeteksi apakah *incremental refresh* dimungkinkan untuk sebuah MQT tersebut, jika tidak, maka mekanisme *full refresh* yang akan dijalankan.

2.6.4.1 *Full Refresh* MQT

Metode ini dilakukan dengan cara mengkomputasi ulang seluruh data pada tabel induk. Pendekatan sederhana adalah dengan menghapus seluruh data dalam

MQT, selanjutnya menghitung ulang data pada tabel induk, kemudian memasukkan ulang data hasil perhitungan kembali kedalam MQT. Pengguna dapat melakukan *complete refresh* kapan saja setelah MQT dibentuk. Mekanisme ini bekerja dengan cara mengeksekusi ulang *query* yang digunakan untuk mendefinisikan MQT. Proses *complete refresh* bisa sangat memakan waktu, terutama ketika *database* harus membaca dan memproses data dalam jumlah yang sangat besar (Lane & Potineni, 2014).

2.6.4.2 *Incremental Refresh* MQT

Incremental refresh mengeliminasi kebutuhan untuk membangun ulang MQT dari dasar. Metode ini hanya memproses data-data yang mengalami perubahan pada tabel induk. Oleh karena itu proses yang dilakukan relatif lebih cepat. MQT dapat di *refresh* sesuai dengan permintaan pengguna atau secara periodik. Dengan kata lain, MQT yang berada dalam *database* yang sama dengan tabel induk dapat di *refresh* kapanpun saat sebuah traksaksi melakukan *commit* perubahannya terhadap tabel induk (Lane & Potineni, 2014).

2.6.4.3 *Refreh Immediate vs Deferred*

MQT yang menggunakan REFRESH IMMEDIATE dapat mempengaruhi performa *query* sama seperti *indexes* bekerja. Secara detail dapat dijelaskan sebagai berikut:

- Mempercepat performa dari SELECT ketika ingin mendapatkan data terbaru.
- Secara otomatis dipilih oleh *optimizer* ketika ditemukan kecocokan dengan *query* yang sedang dijalankan.
- Dapat mengurangi performa dari operasi *insert, update* dan *delete*.
- Tidak dapat dilakukan perubahan (*update*) secara langsung kedalam MQT.
- Terdapat kemungkinan menyebabkan *deadlock* saat dilakukan operasi terhadap tabel-tabel induk yang terkait.

Lain halnya dengan MQT yang menggunakan REFRESH DEFERRED yang sama sekali tidak memberikan pengaruh apapun terhadap performa operasi *insert, update, delete*. Namun data didalamnya akan usang ketika terjadi perubahan pada tabel induk dan tidak segera dilakukan pemutakhiran (*refresh*) terhadap MQT ini (Sanders, 2009).

Tipe pemutakhiran yang digunakan pada penelitian ini adalah *incremental refresh* dengan *refresh deferred* karena proses pemutakhiran MQT dilakukan secara berkala tergantung dari permintaan pengguna.

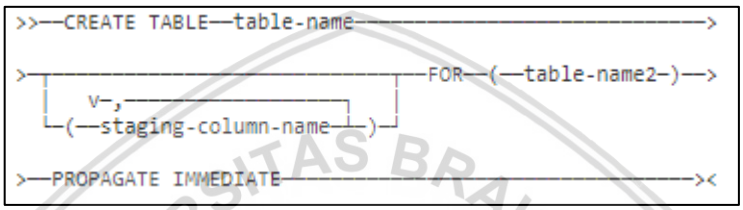
2.7 MQT Staging Table

Staging table merupakan sebuah tabel yang terhubung kepada sebuah MQT. Tabel ini bertugas untuk mencatat seluruh perubahan data pada table-tabel induk

yang terhubung dengan MQT. Pada DBMS *staging table* merupakan bagian dari *temporal data management* karena tabel ini berhubungan dengan pencacatan historis dari perubahan data yang ada pada sebuah tabel tertentu. Catatan perubahan yang ada didalam *staging table* akan digunakan untuk melakukan pemutakhiran MQT secara *incremental* tanpa harus mengkalkulasi seluruh data dari awal. Setelah MQT di *refresh*, seluruh data pada *staging table* akan secara otomatis terhapus (Melnyk, 2005).

2.7.1 Pembuatan Staging Table

Pembuatan *staging table* yang terhubung ke sebuah MQT hampir sama seperti pembuatan tabel normal menggunakan klausa CREATE TABLE.



Gambar 2.23 Definisi sintak untuk pembuatan staging table

Adapun beberapa hal yang perlu diperhatikan ketika mendefinisikan sebuah *staging table* antara lain adalah (Hilker, 2013):

- *table-name*
  Sintak ini berisikan nama dari *staging table* yang akan dibuat.
- *staging-column-name*
  Sintak ini berisikan definisi dari masing-masing kolom yang dimiliki oleh *staging table* yang bersifat opsional, jika tidak didefinisikan maka sistem akan secara otomatis membuat kolom-kolom yang sesuai dengan kebutuhan *refresh* dari MQT yang bersangkutan.
- *FOR table-name2*
  Sintak ini berisikan nama dari MQT terkait yang akan menggunakan *staging table* untuk proses *incremental refresh.* Pada saat pembuatan *staging table,* MQT harus sudah dibuat terlebih dahulu didalam sistem.
- *PROPAGATE IMMEDIATE*
  Sintak ini berfungsi untuk memberitahu sistem bahwa segala perubahan meliputi penambahan, perubahan, penghapusan data akan diteruskan untuk disimpan pada *staging table* yang dibuat. Perlu diperhatikan bahwa ketika *staging table* telah dibuat menggunakan sintak ini, bisa terjadi *error* pada saat operasi perubahan data pada tabel induk. Hal ini biasanya dikarenakan oleh *inconsistent state* atau status dari *staging table* ataupun MQT yang sedang tidak konsisten. Untuk menyelesaikan permasalahan tersebut hanya perlu melakukan SET INTEGRITY dengan atribut IMMEDIATE UNCHECKED pada *staging table* dan MQT yang terkait pada tabel induk tersebut agar dapat digunakan.

2.7.2 Syarat dan Ketentuan Staging Table

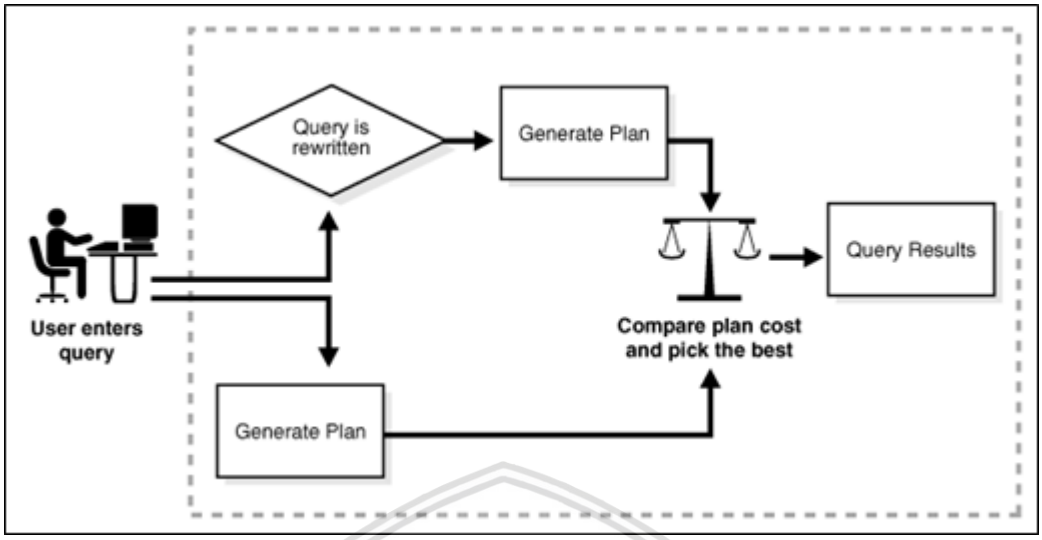
Adapun beberapa hal yang harus diperhatikan ketika akan membuat sebuah *staging table* yang terhubung dengan suatu MQT antara lain (Hilker, 2013):

- Jika kolom-kolom *staging table* didefinisikan, maka harus ada 2 (dua) kolom tambahan dari kolom-kolom yang sudah ada pada MQT terkait.
- Jika sintak *query* pembentuk MQT tidak memiliki klausa GROUP BY, maka harus ada 3 (tiga) kolom tambahan dari kolom-kolom yang sudah ada pada MQT terkait.
- Jika kolom-kolom *staging table* tidak didefinisikan secara eksplisit, maka sistem secara otomatis akan membuat kolom-kolom yang sesuai dengan MQT yang terkait.
- Kolom-kolom tambahan yang akan dibuat secara otomatis pada saat *staging table* dibuat antara lain adalah:
    - GLOBALTRANSID CHAR(8) FOR BIT DATA
      Yaitu kolom yang menyimpan data *transaction ID* dari masing-masing baris data hasil propagasi dari tabel induk
    - GLOBALTRANSTIME CHAR(13) FOR BIT DATA
      Yaitu kolom yang menyimpan data waktu pada saat suatu baris di propagasi dari tabel induk
    - OPERATIONTYPE INTEGER
      Yaitu kolom yang mengidentifikasi jenis transaksi termasuk kedalam operasi *insert, update,* atau *delete* dari baris yang di propagasi dari tabel induk. Kolom ini hanya akan dibuat oleh sistem ketika diperlukan sesuai dengan syarat pembentukan *staging table*.

2.8 Query Rewrite

Penggunaan MQT dimaksutkan untuk meningkatkan kecepatan pemrosesan query terutama pada *database* yang sangat besar. *Query* pada database yang besar sering kali melibatkan proses penggabungan antar tabel-tabel yang saling berelasi, proses agreasi seperti penjumlahan, pengurangan, rata-rata. Operasi tersebut sangat memakan sumber daya dilihat dari segi waktu dan juga kekuatan pemrosesan data yang dibutuhkan. Disinilah MQT memiliki peran yang sangat penting (Melnyk, 2005).

MQT meningkatkan performa *query* dengan melakukan *precalculation* terhadap operasi *joins* dan agregasi kemudian menyimpan hasilnya kedalam *database*. DBMS memiliki sebuah mekanisme yang disebut dengan *query optimizer*, mekanisme ini bertugas untuk menentukan algoritme yang paling optimal ketika ada *statement query* yang dijalankan. D (Hilker, 2013)alam hal ini *query optimizer* juga akan secara otomatis mendeteksi jika didalam *database* sudah terdapat MQT yang berhubungan dengan *statement query* yang sedang dijalankan kemudian menentukan apakan MQT tersebut dapat digunakan atau tidak. Jika ternyata ditemukan MQT yang berhubungan dan dapat digunakan, *query optimizer* akan melakukan penulisan ulang *query (query-rewrite)* dengan secara otomatis memindahkan permintaan data dari tabel biasa ke MQT. Secara umum, *query rewrite* dengan menggunakan MQT dapat meningkatkan performa dari pemrosesan *query*.



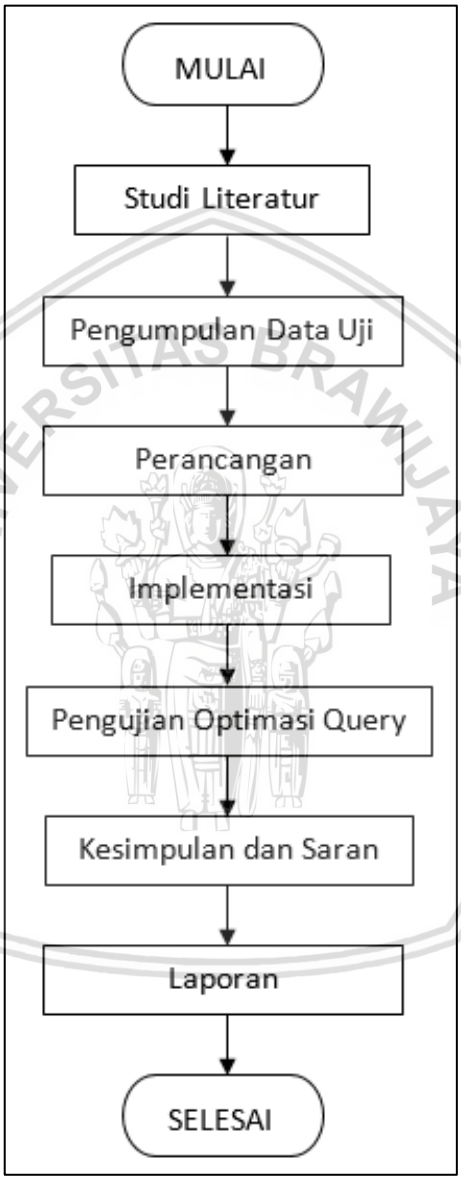
Gambar 2.24 Transparent Query Rewrite (Lane & Potineni, 2014)

Ketika menggunakan *query rewrite*, *query optimizer* akan membuat algoritme utama dengan menggunakan tabel yang diminta oleh pengguna dan sekaligus membuat algoritme dengan menggunakan MQT. Performa dari kedua algoritme tersebut akan dibandingkan untuk kemudian dipilih yang terbaik seperti ilustrasi pada gambar 2.24.

BAB 3 METODOLOGI

3.1 Metode Penelitian

Bab ini berisikan tentang alur metode yang dilakukan dari awal sampai akhir dalam penelitian. Secara lebih jelasnya seperti dapat dilihat pada gambar 3.1.1.



Gambar 3.1 Alur Metodologi Penelitian

3.1.1 Studi Literatur

Tahap studi literatur merupakan tahap pengumpulan referensi dari buku, *e-book*, *website* atau jurnal untuk memperoleh penjelasan tentang dasar-dasar teori yang mendukung penelitian. Dari hasil studi literatur yang dilakukan, terdapat beberapa teori yang mendukung penelitian ini, antara lain:

1. Penelitian yang berkaitan sebelumnya.
2. *Database Sytem*.
3. *Database Management System*.
4. Aljabar Relasional.
5. Temporal Data Management
6. *Materialized Query Table* (MQT).
7. *Staging Table* pada MQT.

3.1.2 Pengumpulan Data Uji

Penelitian ini mengadaptasi permasalahan yang ada pada Sistem Informasi Akademik Mahasiswa Universitas Brawijaya dengan didasarkan pada informasi yang didapat melalui metode pengumpulan data yaitu observasi. Seperti yang dijelaskan pada bagian batasan masalah bahwa data uji yang digunakan pada penelitian ini adalah data mahasiswa Universitas Brawijaya yang didapatkan dari penelitian yang telah dilakukan sebelumnya dan dapat dipertanggungjawabkan. Hal ini didasarkan pada kebutuhan atas jumlah data yang besar dan kemungkinan analisis data yang luas sehingga diputuskan bahwa data mahasiswa tersebut layak untuk dijadikan sebagai data uji penelitian. Adapun jenis data uji yang digunakan terdiri dari antara lain:

- Data Asli yaitu data-data yang didapatkan dari penelitian yang sudah dilakukan sebelumnya.
- Data *Dummy* yaitu data-data yang dihasilkan dari proses *generate* oleh sistem untuk melengkapi kebutuhan penelitian.

3.1.3 Perancangan

Terdapat beberapa tahapan perancangan didalam penelitian ini antara lain adalah sebagai berikut:

1. Rancangan Arsitektur
   Memodelkan rancangan arsitektur sistem yang akan digunakan didalam penelitian.
2. Rancangan Basis Data OLTP
   Memodelkan rancangan sistem basis data yang didalamnya terdapat table-tabel serta hubungan antar table tersebut dengan menggunakan bantuan *Entity Relational Diagram* (ERD).
3. Rancangan Data Penelitian
   Mengumpulkan seluruh data yang didapatkan dari penelitian sebelumnya serta mempersiapkan pembuatan data *dummy* untuk mendukung kebutuhan penelitian.

4. Rancangan MQT dan *Staging Table*
   Memodelkan rancangan MQT yang dibangun berdasarkan hasil dari pengumpulan data dan dilakukan dengan pendekatan struktural. Adapun tahapan yang dilakukan dalam proses pemodelan ini antara lain:

   - Pemodelan menggunakan ERD
     Menggunakan *Entity Relationship Diagram* (ERD) untuk menggambarkan konsep hubungan antara MQT, *staging table,* dan table-tabel induk yang berkaitan untuk memperjelas bagaimana data serta perubahan (delta) data diambil, dimodifikasi, dan kemudian dimasukkan kedalam MQT sesuai dengan kebutuhan.

   - Pemodelan menggunakan aljabar relasional
     Setelah ERD selesai dibentuk maka akan dilanjutkan dengan proses membuat rancangan *query* dengan menggunakan notasi aljabar relasional. Hal ini bertujuan untuk memudahkan proses pembentukan *query* dengan mengacu pada aljabar yang telah dirancang.

5. Rancangan *Refresh* MQT
   Membuat skenario *full refresh* dan *incremental refresh* memanfaatkan *staging table* untuk melihat perbedaan performa sistem.

3.1.4 Implementasi

Implementasi sistem akan didasarkan pada hasil rancangan yang telah dibuat pada proses perancangan sistem. Sehingga tahapan-tahapan implementasi dapat dikatakan hampir sama dengan tahapan-tahapan di perancangan yaitu :

1. Implementasi Arsitektur
   Arsitektur sistem akan diimplementasikan menggunakan teknologi virtualisasi. Dengan menggunakan perangkat lunak *VMWare*.
2. Implementasi Basis Data OLTP
   Implementasi dilakukan dengan mengeksekusi DDL (*Data Definition Languae*) dari skema basis data ke *server* basis data OLTP. Pada implementasi ini penulis menggunakan sistem basis data IBM DB2.
3. Pembuatan Data Penelitian
   Merupakan proses penyimpanan data ke basis data OLTP. Proses pembuatan menggunakan bantuan program *Ms Office Excel* dengan fasilitas *import* yang dimiliki DBMS IBM DB2.
4. Implementasi MQT dan *Staging Table*
   Proses implementasi MQT dilakukan dengan menerjemahkan aljabar relasional yang telah dibuat menjadi SQL (*Structured Query Language*). Dan melakukan *deploy* ke *database*. Kemudian dilanjutkan dengan pembuatan *staging table* yang dihubungkan dengan masing-masing MQT sebagai syarat utama dilakukannya *incremental refresh.*

5. Proses *Refresh* MQT
   Proses implementasi *full refresh* dan *incremental refresh* memanfaatkan *staging table*. Setelah itu dilakukan percobaan eksekusi *query* pada keduanya untuk melihat perbedaan performa sistem.

3.1.5 Pengujian Optimasi Query

Pengujian sistem dilakukan untuk menguji dampak penggunaan *incremental refresh* dibandingkan dengan metode *full refresh* pada *materialized query table* terhadap performa *query*. Pengujian dilakukan dengan menggunakan metode pengujian performa (*performance test*). Disini akan dilihat seberapa signifikan perubahan yang diberikan terhadap waktu eksekusi *query* dan besarnya sumber daya yang dibutuhkan pada saat proses pemutakhiran data yang ada didalam MQT.

3.1.5.1 *Pengujian Performa (Performance Test)*

Pengujian ini memiliki skenario dengan melakukan eksekusi *query* terhadap MQT dengan mekanisme *full refresh* serta *incremental refresh* pada MQT untuk mengetahui berapa lama waktu yang dibutuhkan untuk memprosesnya dan juga melihat berapa banyak sumber daya CPU serta *input output* (I/O) yang diperlukan oleh masing-masing mekanisme *refresh*. Kemudian hasil dari keduanya akan dibandingkan untuk mengetahui adanya peningkatan atau tidak.

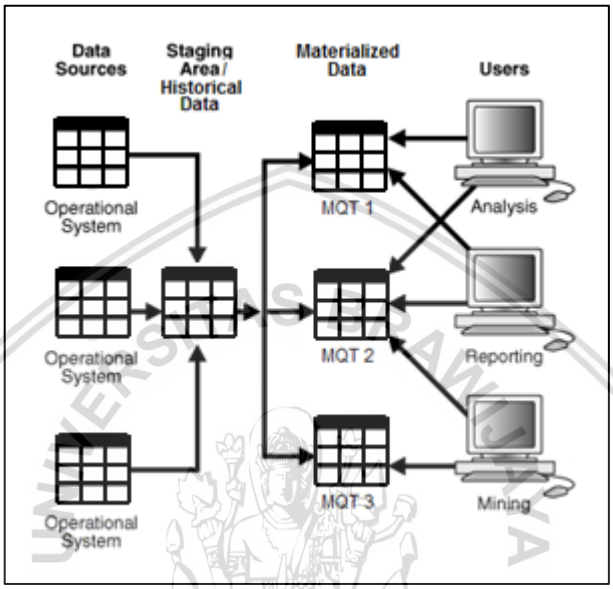
3.1.6 Penyusunan Laporan

Pada tahap akhir yang dilakukan pada penelitian adalah penyusunan hasil penelitian dalam bentuk laporan. Laporan akhir ini berupa dokumentasi dari tahap awal hingga akhir pada penelitian yang dilakukan. Hasil penelitian disusun dan dijabarkan sesuai dengan kaidah penulisan ilmiah yang benar.

BAB 4 PERANCANGAN

4.1 Perancangan Arsitektur

Perancangan arsitektur MQT pada penelitian ini melibatkan beberapa komponen utama yaitu sumber data (*source*), data *staging* (*historical change*), *materialized data*, dan juga pengguna akhir (*user*).



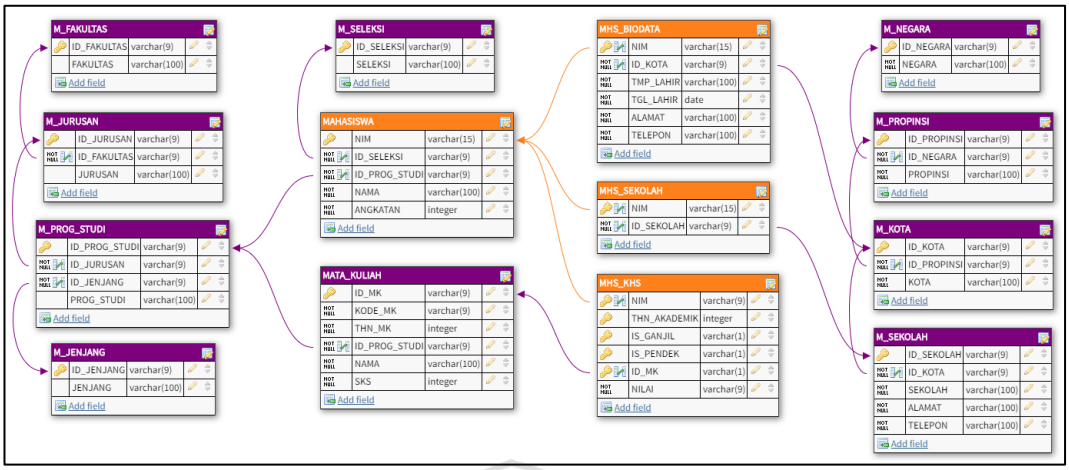
Gambar 4.1 Rancangan Arsitektur MQT

Proses perancangan arsitektur seperti yang ditunjukan oleh Gambar 4.1 adalah sebagai berikut:

1. Penyediaan data sumber (*source*) yang diperoleh dari penelitian sebelumnya.
2. Proses pencataan (*capture*) dari setiap perubahan yang terjadi pada data sumber dan disimpan dalam *staging table*.
3. Implementasi *Materialize Query Table* (MQT) sebagai pembentukan data analitik yang dapat dimanfaatkan oleh pengguna.
4. Hasil data dari MQT dapat digunakan sebagai bahan analisis, pelaporan maupun sebagai data sumber untuk kebutuhan *data mining*.

4.2 Perancangan Basis Data

Basis data yang digunakan dalam penelitian ini adalah hasil adaptasi dari penelitian sebelumnya yaitu sistem basis data SIAKAD pada Universitas Brawijaya (Nurcahyo, 2015).



Gambar 4.2 Diagram Rancangan Basis Data

Gambar 4.2 merupakan rancangan ERD pada sistem basis data SIAKAD yang telah dimodifikasi untuk menyesuaikan kebutuhan penelitian ini. Didalamnya terdapat beberapa tabel dengan rincian sebagai berikut:

Tabel 4.1 Tabel-Tabel dalam Basis Data

Nama Tabel	Deskripsi
M_SELEKSI	Merupakan tabel yang menyimpan data master jalur seleksi mahasiswa
M_JENJANG	Merupakan tabel yang menyimpan data master jenjang pendidikan mahasiswa
M_FAKULTAS	Merupakan tabel yang meyimpan data master fakultas yang dimiliki oleh universitas
M_JURUSAN	Merupakan tabel yang meyimpan data master jurusan yang dimiliki oleh universitas
M_PROG_STUDI	Merupakan tabel yang meyimpan data master program studi yang dimiliki oleh universitas
M_NEGARA	Merupakan tabel yang menyimpan data master negara (dalam penelitian ini hanya menggunakan negara Indonesia)
M_PROPINSI	Merupakan tabel yang menyimpan data master propinsi yang ada di Indonesia
M_KOTA	Merupakan tabel yang menyimpan data master kota yang ada di Indonesia
M_SEKOLAH	Merupakan tabel yang menyimpan data master sekolah yang ada di Indonesia
MAHASISWA	Merupakan tabel yang menyimpan data mahasiswa beserta jalur seleksi, jenjang, fakultas, jurusan, program studi, dan angkatan yang dimilikinya
MATA_KULIAH	Merupakan tabel yang menyimpan data master mata kuliah yang dapat di ampu oleh mahasiswa

MHS_BIODATA	Merupakan tabel yang menyimpan detail biodata mahasiswa termasuk alamat asal dan nomor kontak
MHS_SEKOLAH	Merupakan tabel yang menyimpan detail sekolah asal dari mahasiswa
MHS_KHS	Merupakan tabel yang menyimpan hasil akademik yang telah ditempuh oleh mahasiswa

4.3 Perancangan Data Penelitian

Data yang digunakan bukanlah data yang valid dan benar. Hal ini didasari oleh alasan bahwa kebenaran data bukan menjadi syarat yang harus terpenuhi didalam penelitian ini. Disamping itu juga dikarenakan data yang digunakan bersifat rahasia dan tidak dapat disebarluaskan. Oleh sebab itu data yang digunakan dalam penelitian ini terdiri dari data asli dan data *dummy* yang merupakan hasil dari pembuatan otomatis (*generate*) yang dilakukan oleh komputer.

Data Asli yang digunakan pada penelitian ini diperoleh dari penelitian yang telah dilakukan sebelumnya dan dapat dipertanggung jawabkan. Data-data tersebut didapatkan dalam format dokumen (.csv) yang antara lain adalah: Data Seleksi, Data Jenjang, Data Fakultas, Data Jurusan, Data Program Studi, Data Mata Kuliah, Data Sekolah, Data Kota, Data Propinsi, Data Negara. Bentuk, struktur, serta isian untuk data-data tersebut dimodifikasi untuk memenuhi kebutuhan penelitian ini.

Data *dummy* diperoleh dari hasil pembuatan otomatis (*generate*) oleh komputer dengan bantuan progam Microsoft Office Excel. Data-data tersebut meliputi antara lain: Data Mahasiswa, Data Biodata Mahasiswa, Data Sekolah Asal Mahasiswa, serta Data KHS Mahasiswa.

4.4 Perancangan Materialized Query Table

Langkah-langkah dalam perancangan MQT adalah dengan menentukan kemudian membuat rancangan dari sebuah proses *query* data. Seperti yang dijabarkan di bab sebelumnya bahwa MQT merupakan sebuah tabel yang dihasilkan dari proses *query*, pada banyak kasus MQT dimanfaatkan untuk menampung data yang bersifat hasil perhitungan atau agregasi. Hal tersebut dimaksutkan agar ketika suatu bentuk perhitungan data diminta oleh pengguna, DBMS tidak perlu lagi menghitung ulang dari tabel-tabel operasional, melainkan dapat mengakses MQT secara langsung sehingga dapat mempercepat proses memperoleh data. Data-data yang tesimpan didalam MQT biasanya banyak digunakan sebagai bahan analisis ataupun pelaporan.

Berdasarkan hasil konsultasi dengan dosen pembimbing maka telah ditentukan bahwa MQT yang akan dirancang akan dapat mengakomodir kebutuhan data berikut:

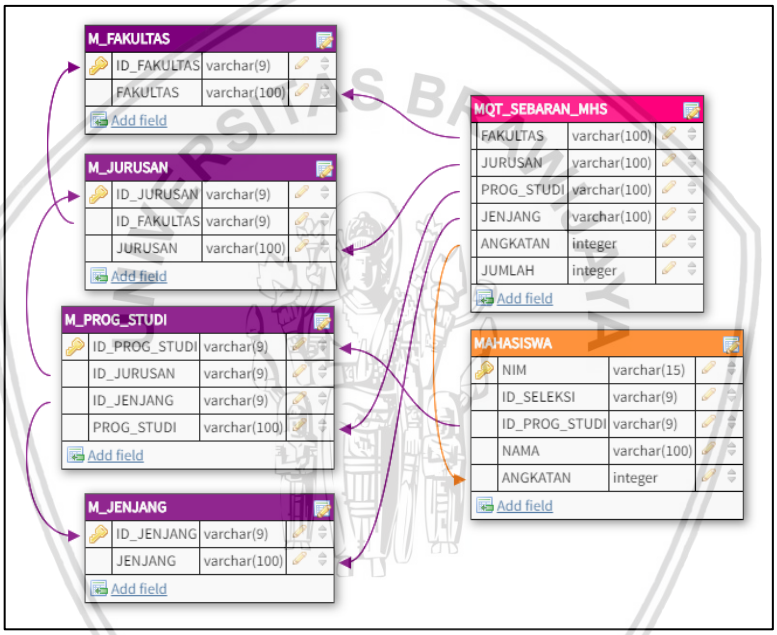
- Data sebaran mahasiswa.
- Data sebaran mahasiswa berdasarkan sekolah asal.

- Data sebaran mahasiswa berdasarkan daerah asal.

Setelah menentukan MQT apa saja yang akan dibangun makan langkah selanjutnya adalah merancang algoritme *query* agar data yang ditampung sesuai dengan ketentuan. Perancangan algoritme MQT ini akan dilakukan dengan menggunakan aljabar relasional.

4.4.1 MQT Sebaran Mahasiswa

MQT ini berfungsi untuk menyimpan data perhitungan sebaran mahasiswa. Dimensi perhitungan yang digunakan antara lain adalah fakultas, jurusan, jenjang, program studi, dan angkatan. Untuk itu tabel-tabel yang diperlukan antara lain adalah MAHASISWA, M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS. Lebih jelasnya seperti yang ditunjukkan pada Gambar 4.3.



Gambar 4.3 Diagram Rancangan MQT Sebaran Mahasiswa

Untuk membantu memudahkan proses implementasi MQT maka perlu dibuat algoritme secara lebih detail untuk pembangunan *query*. Berikut merupakan notasi aljabar relasional untuk membentuk MQT Sebaran Mahasiswa:

1. π M_FAKULTAS.FAKULTAS, M_JURUSAN.JURUSAN, M_JENJANG.JENJANG,
2. M_PROG_STUDI.PROG_STUDI, MAHASISWA.ANGKATAN, JUMLAH
3. γ M_FAKULTAS.FAKULTAS, M_JURUSAN.JURUSAN, M_JENJANG.JENJANG,
4. M_PROG_STUDI.PROG_STUDI, MAHASISWA.ANGKATAN; COUNT(MAHASISWA.NIM) $\rightarrow$JUMLAH
5. σ M_PROG_STUDI.ID_PROG_STUDI = MAHASISWA.ID_PROG_STUDI and
6. M_JENJANG.ID_JENJANG = M_PROG_STUDI.ID_JENJANG and
7. M_JURUSAN.ID_JURUSAN = M_PROG_STUDI.ID_JURUSAN and
8. M_FAKULTAS.ID_FAKULTAS = M_JURUSAN.ID_FAKULTAS (MAHASISWA $\times$ M_PROG_STUDI $\times$ M_JENJANG
9. $\times$ M_JURUSAN $\times$ M_FAKULTAS)

Gambar 4.4 Aljabar Relasional MQT Sebaran Mahasiswa

Penjelasan dari kode aljabar pada Gambar 4.4 adalah sebagai berikut:

Baris 1-2 : Proses *projection* dari kolom-kolom hasil penggabungan tabel dan *aggregate* yang nantinya akan menjadi kolom-kolom dari MQT.

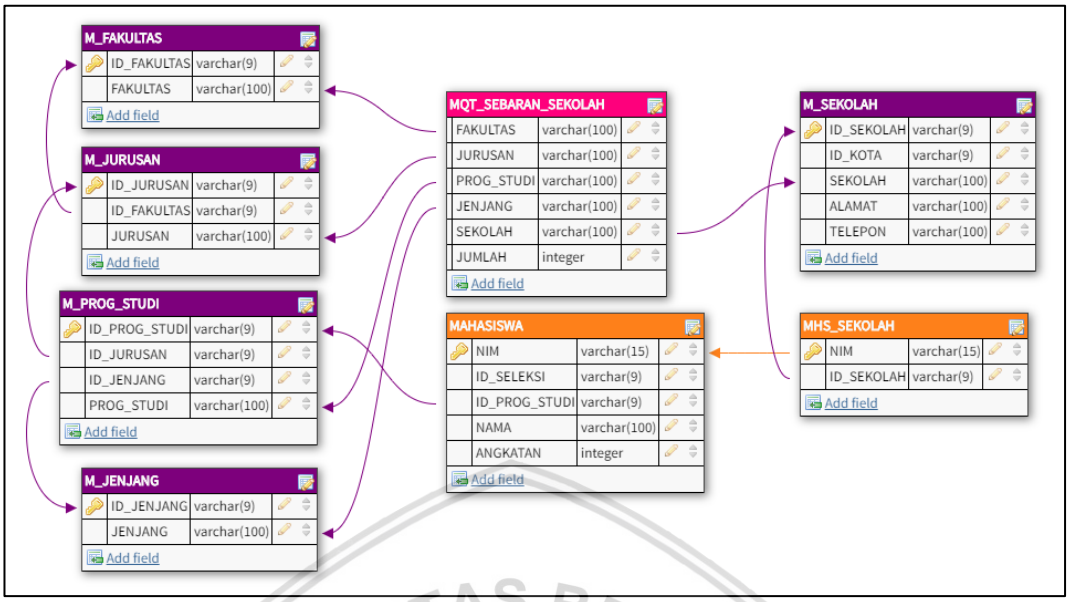
Baris 3-4 : Proses *aggregate count* pada data NIM di tabel MAHASISWA untuk kemudian di *rename* menjadi kolom JUMLAH berdasarkan *grouping* kolom-kolom yang terkait.

Baris 5-8 : Proses pendefinisian seleksi kondisi yang menyatakan hubungan antar relasi yang dinyatakan dalam kolom-kolom sebagai syarat untuk melakukan penggabungan.

Baris 8 : Proses penggabungan relasi MAHASISWA, M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS dengan seleksi kondisi yang telah didefinisikan sebelumnya.

4.4.2 MQT Sebaran Mahasiswa berdasarkan Sekolah Asal

MQT ini berfungsi untuk menyimpan data perhitungan sebaran mahasiswa berdasarkan dari sekolah asal. Dimensi perhitungan yang digunakan antara lain adalah fakultas, jurusan, jenjang, program studi, dan sekolah asal. Untuk itu tabel-tabel yang diperlukan antara lain adalah MHS_SEKOLAH, M_SEKOLAH, MAHASISWA, M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS. Lebih jelasnya seperti yang ditunjukkan pada Gambar 4.5.



Gambar 4.5 Diagram Rancangan MQT Sebaran Mahasiswa berdasarkan Sekolah Asal

Untuk membantu memudahkan proses implementasi MQT maka perlu dibuat algoritme secara lebih detail untuk pembangunan *query*. Berikut merupakan notasi aljabar relasional untuk membentuk MQT Sebaran Mahasiswa berdasarkan Sekolah Asal:

1. π M_FAKULTAS.FAKULTAS, M_JURUSAN.JURUSAN, M_JENJANG.JENJANG,
2. M_PROG_STUDI.PROG_STUDI, M_SEKOLAH.SEKOLAH, JUMLAH
3. γ M_FAKULTAS.FAKULTAS, M_JURUSAN.JURUSAN, M_JENJANG.JENJANG,
4. M_PROG_STUDI.PROG_STUDI, M_SEKOLAH.SEKOLAH; COUNT(MAHASISWA.NIM) $\rightarrow$JUMLAH
5. σ M_SEKOLAH.ID_SEKOLAH = MHS_SEKOLAH.ID_SEKOLAH and
6. MAHASISWA.NIM = MHS_SEKOLAH.NIM and
7. M_PROG_STUDI.ID_PROG_STUDI = MAHASISWA.ID_PROG_STUDI and
8. M_JENJANG.ID_JENJANG = M_PROG_STUDI.ID_JENJANG and
9. M_JURUSAN.ID_JURUSAN = M_PROG_STUDI.ID_JURUSAN and
10. M_FAKULTAS.ID_FAKULTAS = M_JURUSAN.ID_FAKULTAS $^{(MHS\_SEKOLAH \times M\_SEKOLAH \times MAHASISWA}$
11. $\times$ M_PROG_STUDI $\times$ M_JENJANG $\times$ M_JURUSAN $\times$ M_FAKULTAS)

Gambar 4.6 Aljabar Relasional MQT Sebaran Mahasiswa berdasarkan Sekolah Asal

Penjelasan dari kode aljabar pada Gambar 4.6 adalah sebagai berikut:

Baris 1-2 : Proses *projection* dari kolom-kolom hasil penggabungan tabel dan *aggregate* yang nantinya akan menjadi kolom-kolom dari MQT.

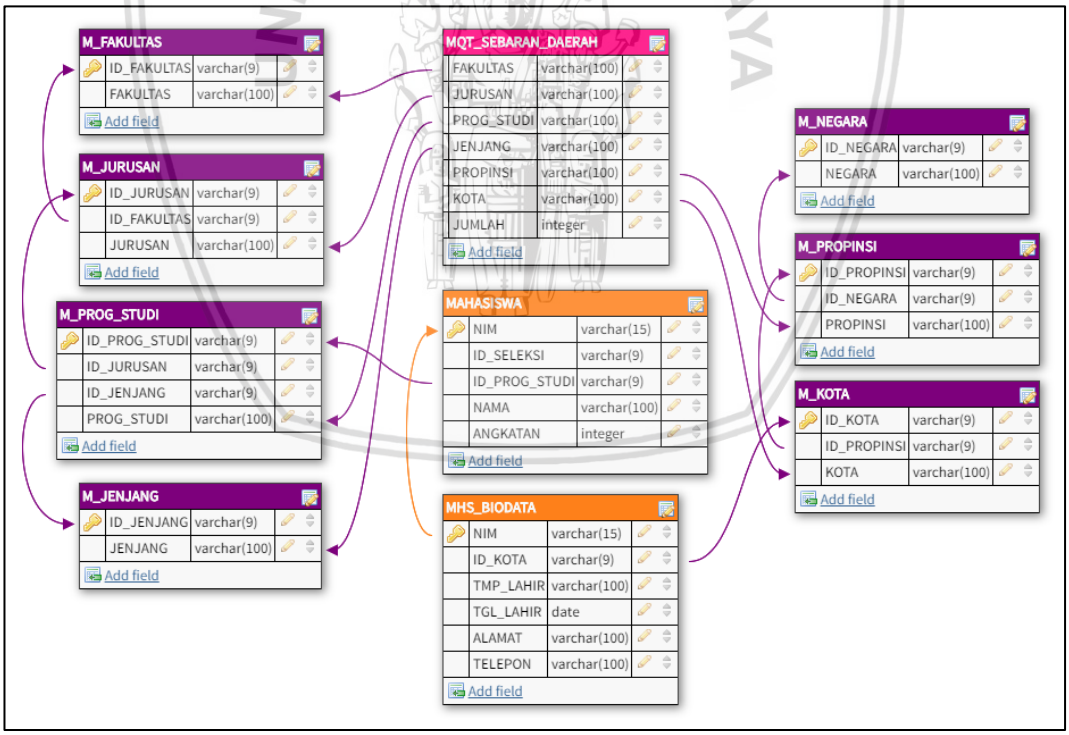
Baris 3-4 : Proses *aggregate count* pada data NIM di tabel MAHASISWA untuk kemudian di *rename* menjadi kolom JUMLAH berdasarkan *grouping* kolom-kolom yang terkait.

Baris 5-10 : Proses pendefinisian seleksi kondisi yang menyatakan hubungan antar relasi yang dinyatakan dalam kolom-kolom sebagai syarat untuk melakukan penggabungan.

Baris 10-11 : Proses penggabungan relasi MHS_SEKOLAH, M_SEKOLAH, MAHASISWA, M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS dengan seleksi kondisi yang telah didefinisikan sebelumnya.

4.4.3 MQT Sebaran Mahasiswa berdasarkan Daerah Asal

MQT ini berfungsi untuk menyimpan data perhitungan sebaran mahasiswa berdasarkan dari daerah asal. Dimensi perhitungan yang digunakan antara lain adalah fakultas, jurusan, jenjang, program studi, propinsi dan kota. Untuk itu tabel-tabel yang diperlukan antara lain adalah MHS_BIODATA, M_KOTA, M_PROPINSI, MAHASISWA, M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS. Lebih jelasnya seperti yang ditunjukkan pada Gambar 4.7.



Gambar 4.7 Diagram Rancangan MQT Sebaran Mahasiswa berdasarkan Daerah Asal

Untuk membantu memudahkan proses implementasi MQT maka perlu dibuat algoritme secara lebih detail untuk pembangunan *query*. Berikut merupakan

notasi aljabar relasional untuk membentuk MQT Sebaran Mahasiswa berdasarkan Sekolah Asal:

```
1.  π M_FAKULTAS.FAKULTAS, M_JURUSAN.JURUSAN, M_JENJANG.JENJANG,
2.      M_PROG_STUDI.PROG_STUDI, M_PROPINSI.PROPINSI, M_KOTA.KOTA, JUMLAH
3.    γ M_FAKULTAS.FAKULTAS, M_JURUSAN.JURUSAN, M_JENJANG.JENJANG,
4.        M_PROG_STUDI.PROG_STUDI, M_PROPINSI.PROPINSI, M_KOTA.KOTA;
5.        COUNT(MAHASISWA.NIM)→JUMLAH
6.  σ MAHASISWA.NIM = MHS_BIODATA.NIM and
7.    M_PROG_STUDI.ID_PROG_STUDI = MAHASISWA.ID_PROG_STUDI and
8.    M_JENJANG.ID_JENJANG = M_PROG_STUDI.ID_JENJANG and
9.    M_JURUSAN.ID_JURUSAN = M_PROG_STUDI.ID_JURUSAN and
10.   M_FAKULTAS.ID_FAKULTAS = M_JURUSAN.ID_FAKULTAS and
11.   M_KOTA.ID_KOTA = MHS_BIODATA.ID_KOTA and
12.   M_PROPINSI.ID_PROPINSI = M_KOTA.ID_PROPINSI (MHS_BIODATA × M_KOTA × M_PROPINSI
13.   × MAHASISWA × M_PROG_STUDI × M_JENJANG × M_JURUSAN × M_FAKULTAS)
```

Gambar 4.8 Aljabar Relasional MQT Sebaran Mahasiswa berdasarkan Daerah Asal

Penjelasan dari kode aljabar pada Gambar 4.8 adalah sebagai berikut:

Baris 1-2 : Proses *projection* dari kolom-kolom hasil penggabungan tabel dan *aggregate* yang nantinya akan menjadi kolom-kolom dari MQT.

Baris 3-5 : Proses *aggregate count* pada data NIM di tabel MAHASISWA untuk kemudian di *rename* menjadi kolom JUMLAH berdasarkan *grouping* kolom-kolom yang terkait.

Baris 6-12 : Proses pendefinisian seleksi kondisi yang menyatakan hubungan antar relasi yang dinyatakan dalam kolom-kolom sebagai syarat untuk melakukan penggabungan.

Baris 12-13 : Proses penggabungan relasi MHS_BIODATA, M_KOTA, M_PROPINSI, MAHASISWA, M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS dengan seleksi kondisi yang telah didefinisikan sebelumnya.

4.5 Perancangan Staging Table

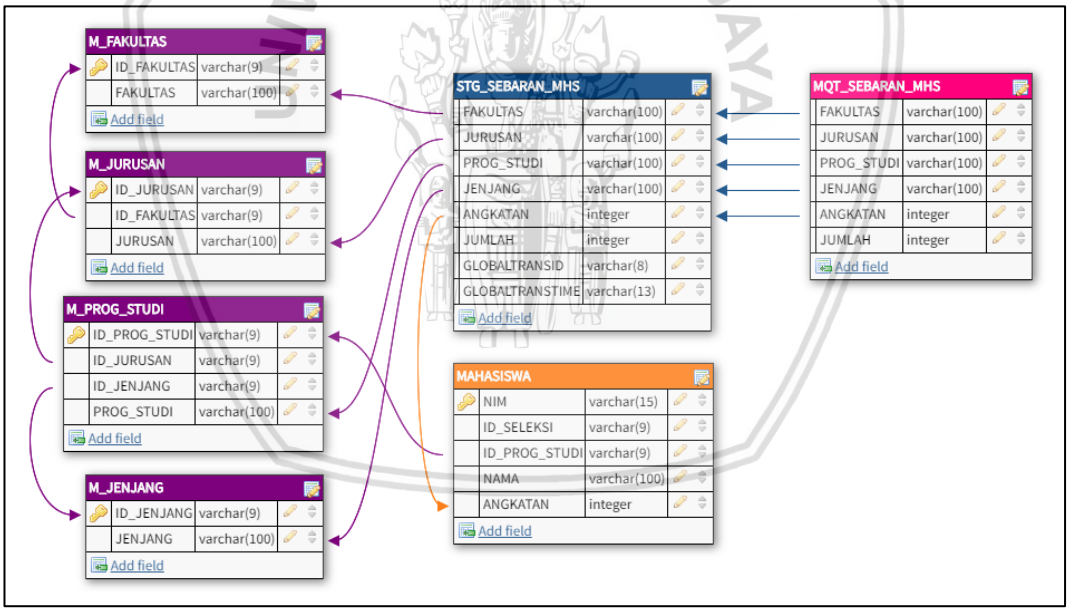
Seperti yang telah dijabarkan pada bab sebelumnya bahwa untuk dapat dilakukan mekanisme *incremental refresh* pada suatu MQT maka ada sebuah syarat yang harus dipenuhi yaitu adanya sebuah *staging table.* Masing-masing MQT akan memiliki setidaknya 1 (satu) *staging table* yang terhubung kepadanya. *Staging table* tersebut berfungsi untuk mencatat seluruh perubahan data yang

terjadi pada seluruh tabel induk yang terhubung ke MQT. Pada saat proses *incremental refresh* dijalankan MQT tidak akan mengumpulkan dan menghitung data dari tabel induk, melainkan dengan mengambil data yang ada didalam *staging table* untuk melakukan pemutakhiran data.

Pada saat pembuatan *staging table* harus diberikan klausa PROPAGATE IMMEDIATE. Hal ini bertujuan untuk memberitahu sistem bahwa segala perubahan meliputi penambahan, perubahan, penghapusan data akan diteruskan untuk disimpan pada *staging table* yang dibuat.

4.5.1 Staging Table Sebaran Mahasiswa

Staging Table ini berfungsi untuk menyimpan seluruh perubahan data pada tabel-tabel yang terhubung ke MQT. Sehingga ketika dilakukan proses *refresh* pada MQT Sebaran Mahasiswa, MQT tersebut tidak akan mengakses tabel-tabel induk dimana data berada, melainkan akan mengakses *staging table* ini untuk melihat apakah telah terjadi perubahan pada tabel induk atau tidak, jika ditemukan perubahan yang terjadi maka selanjutnya data didalam MQT akan diperbarui mengikuti perubahan yang telah tercatat. Lebih jelasnya seperti yang ditunjukkan pada Gambar 4.9.

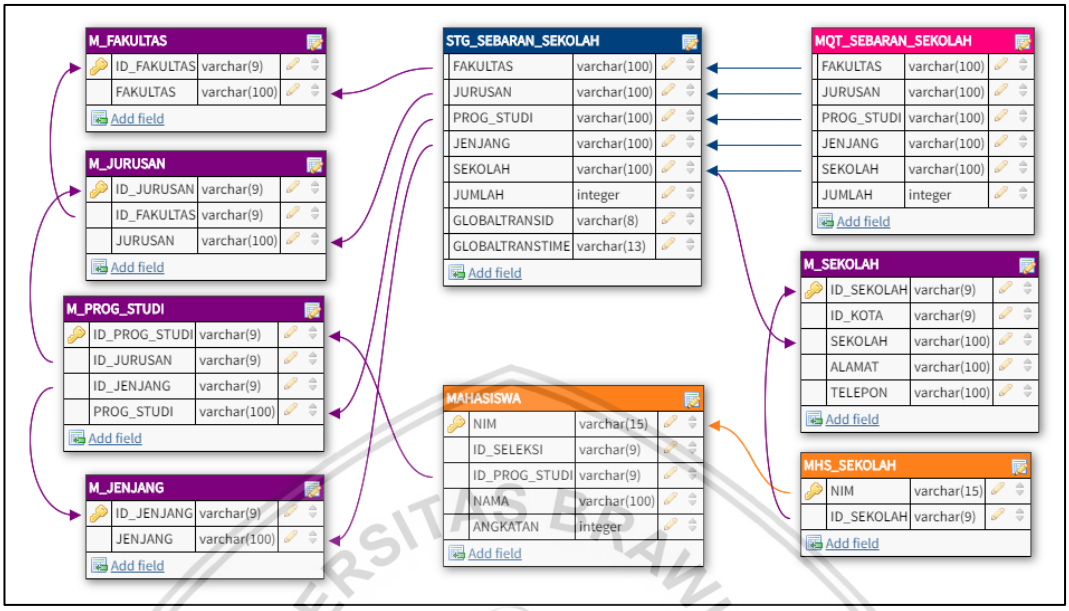


Gambar 4.9 Diagram Rancangan Staging Table Sebaran Mahasiswa

4.5.2 Staging Table Sebaran Mahasiswa berdasarkan Sekolah Asal

Staging Table ini berfungsi untuk menyimpan seluruh perubahan data pada tabel-tabel yang terhubung ke MQT. Sehingga ketika dilakukan proses *refresh* pada MQT Sebaran Mahasiswa berdasarkan Sekolah Asal, MQT tersebut tidak akan mengakses tabel-tabel induk dimana data berada, melainkan akan mengakses *staging table* ini untuk melihat apakah telah terjadi perubahan pada tabel induk atau tidak, jika ditemukan perubahan yang terjadi maka selanjutnya

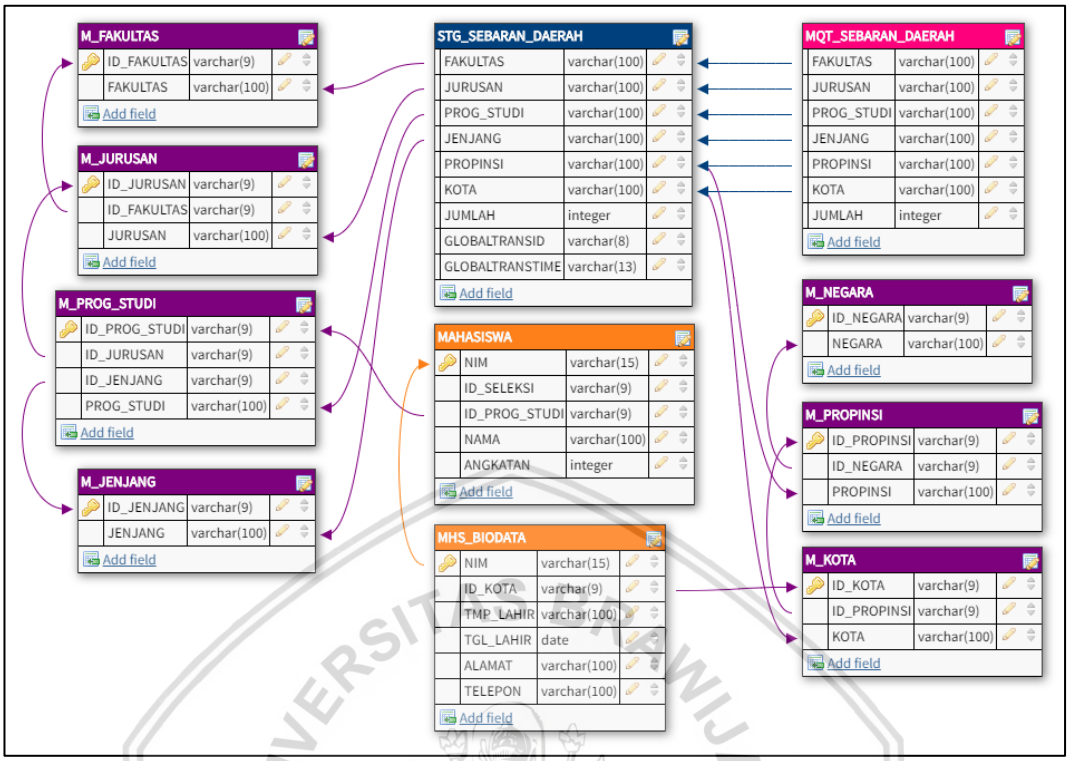
data didalam MQT akan diperbarui mengikuti perubahan yang telah tercatat. Lebih jelasnya seperti yang ditunjukkan pada Gambar 4.10.



Gambar 4.10 Diagram Rancangan Staging Table Sebaran Mahasiswa berdasarkan Sekolah Asal

4.5.3 Staging Table Sebaran Mahasiswa berdasarkan Daerah Asal

Staging Table ini berfungsi untuk menyimpan seluruh perubahan data pada tabel-tabel yang terhubung ke MQT. Sehingga ketika dilakukan proses *refresh* pada MQT Sebaran Mahasiswa berdasarkan Daerah Asal, MQT tersebut tidak akan mengakses tabel-tabel induk dimana data berada, melainkan akan mengakses *staging table* ini untuk melihat apakah telah terjadi perubahan pada tabel induk atau tidak, jika ditemukan perubahan yang terjadi maka selanjutnya data didalam MQT akan diperbarui mengikuti perubahan yang telah tercatat. Lebih jelasnya seperti yang ditunjukkan pada Gambar 4.11.



Gambar 4.11 Diagram Rancangan Staging Table Sebaran Mahasiswa berdasarkan Daerah Asal

BAB 5 IMPLEMENTASI

5.1 Implementasi Arsitektur

Sub-bab ini akan menjelaskan mengenai implementasi arsitektur meliputi lingkungan perangkat keras (*hardware*), lingkungan perangkat lunak (*software*) dan virtualisasi server yang digunakan dalam penelitian. Simulasi penelitian hanya akan berada pada sisi server dikarenakan pembahasan difokuskan pada proses *incremental refresh* MQT yang berjalan pada DBMS di sisi server.

5.1.1 Lingkungan Perangkat Keras

Dalam penelitian ini digunakan perangkat keras pendukung implementasi dengan spesifikasi seperti berikut:

1. Processor Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz 3.40GHz
2. RAM DDR 3 4GB
3. *Harddisk* 500GB
4. Monitor
5. *Keyboard*
6. *Mouse*

5.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan untuk mendukung penelitian ini antara lain:

1. Sistem Operasi Microsoft Windows 10 Pro x64 bit
2. VMWare Workstation 11.1.2 build-2780323
3. IBM Data Studio 4.1.0.0 Client

5.1.3 Virtualisasi Server

Implementasi server yang digunakan dalam penelitian ini memanfaatkan perangkat lunak VMWare dengan spesifikasi:

1. Sistem Operasi CentOS 6.5 x64 bit
2. Prosessor *Dual Core* (dua inti prosessor)
3. RAM 2GB
4. *Harddisk* 40GB
5. DB2 10.5.0.1 Standart Express-C Edition

5.2 Implementasi Basis Data

Implementasi sistem basis data dari rancangan yang telah dibuat pada bab sebelumnya. Dimulai dengan merubah skema / desain rancangan menjadi bentuk

Data Definition Language (DDL) kemudian di eksekusi (*deploy*) pada server yang telah disiapkan. Perintah DDL yang dilakukan dapat dilihat pada lampiran A.

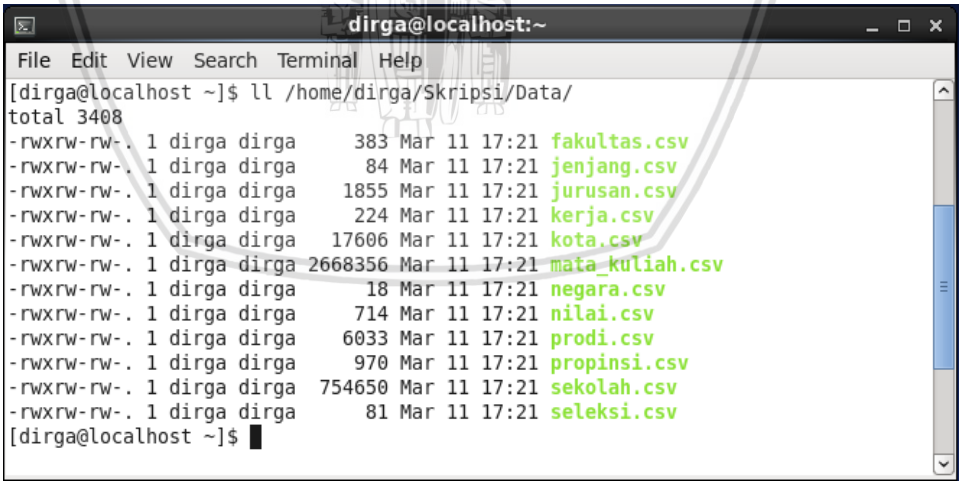
5.3 Implementasi Data Penelitian

Seperti yang telah dijelaskan pada bab sebelumnya bahwa penelitian ini menggunakan 2 (dua) jenis data yaitu Data Asli dan Data *Dummy*. Berikut akan dijelaskan bagaimana kedua data tersebut diimplementasikan.

5.3.1 Data Asli

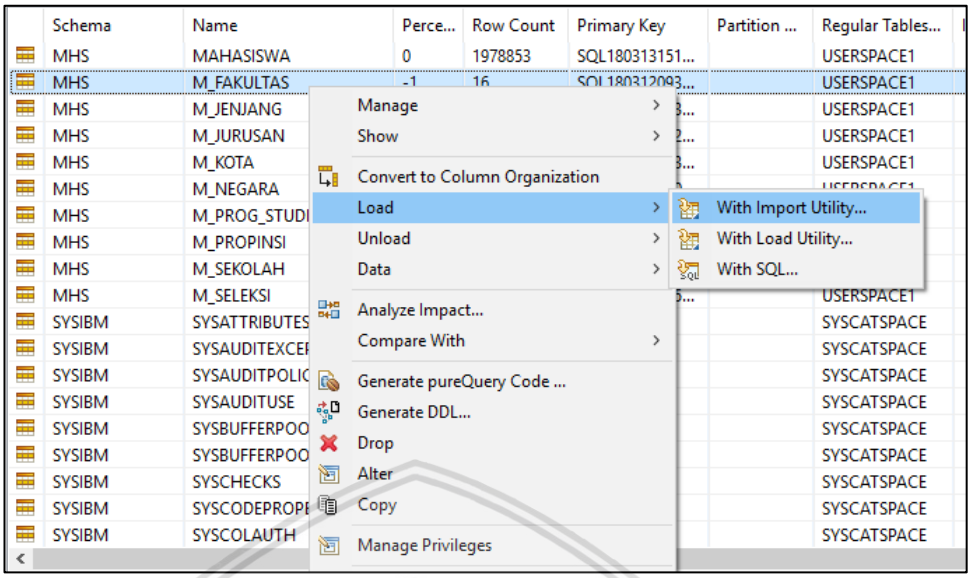
Data Asli yang digunakan pada penelitian ini penulis peroleh dari penelitian yang telah dilakukan sebelumnya dan dapat dipertanggung jawabkan. Data-data tersebut didapatkan dalam format dokumen (.csv) yang antara lain adalah: Data Seleksi, Data Jenjang, Data Fakultas, Data Jurusan, Data Program Studi, Data Mata Kuliah, Data Sekolah, Data Kota, Data Propinsi, Data Negara. Bentuk, struktur, serta isian untuk data-data tersebut dimodifikasi untuk memenuhi kebutuhan penelitian ini. Data tersebut dimasukkan kedalam basis data dengan memanfaatkan fitur *import data* yang telah disediakan oleh IBM Data Studio. Untuk lebih jelasnya dapat dilihat pada langkah-langkah sebagai berikut:

1. Langkah pertama adalah mengumpulkan semua data yang masih dalam format (.csv) pada *folder* tertentu. Kemudian *upload folder* tersebut kedalam salah satu direktori pada *server.* Contohnya pada direktori "*/home/dirga/skripsi/data*" dan pastikan bahwa seluruh dokumen data dapat diakses seperti pada Gambar 5.1.



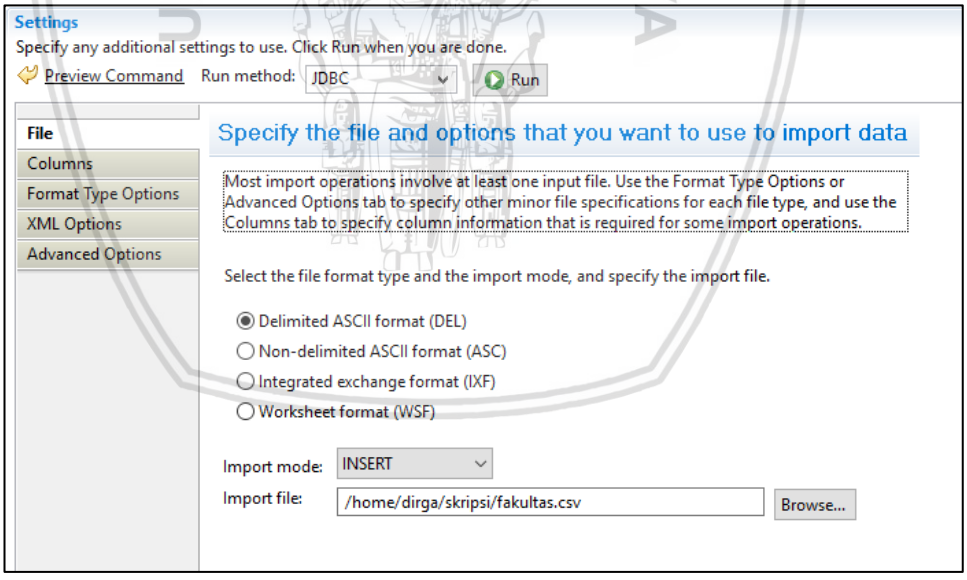
Gambar 5.1 Upload Data ke Server

2. Kemudian langkah selanjutnya adalah memasukkan data tersebut kedalam *table-table* yang sesuai pada basis data. Hal ini dapat dilakukan dengan menggunakan fitur *Import Utility* yang telah disediakan oleh IBM Data Studio seperti yang dapat dilihat pada Gambar 5.2.



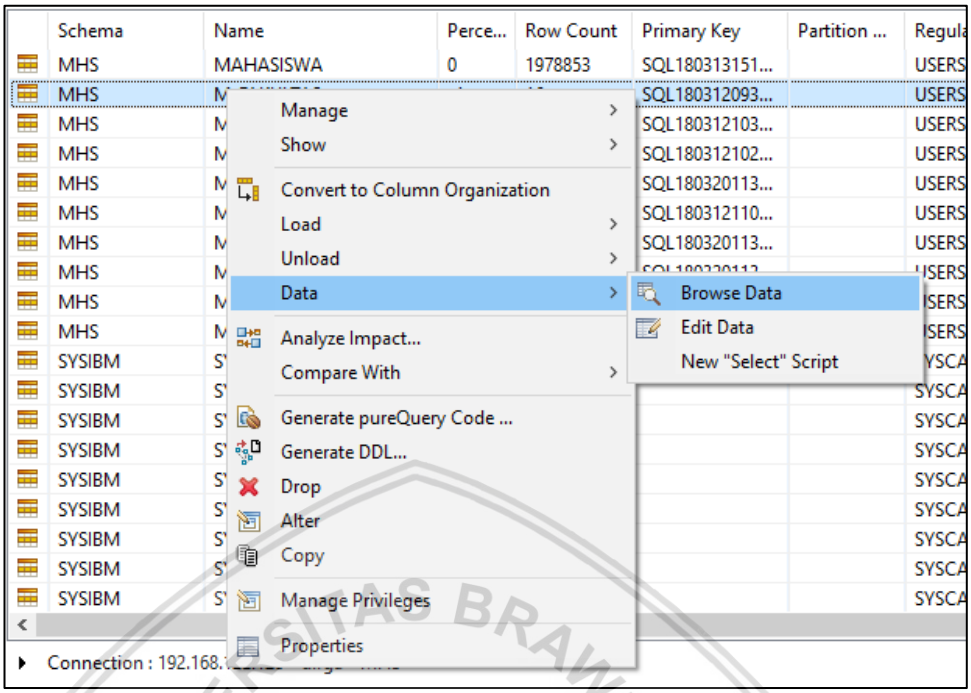
Gambar 5.2 Memasukkan Data dengan Import Utility

3. Setelah muncul jendelan *Import Utility,* selanjutnya adalah memilih tempat dimana dokumen yang terkait berada pada *server* dan jika telah benar maka tekan tombol "*Run*". Lebih jelasnya seperti yang dapat dilihat pada Gambar 5.3.



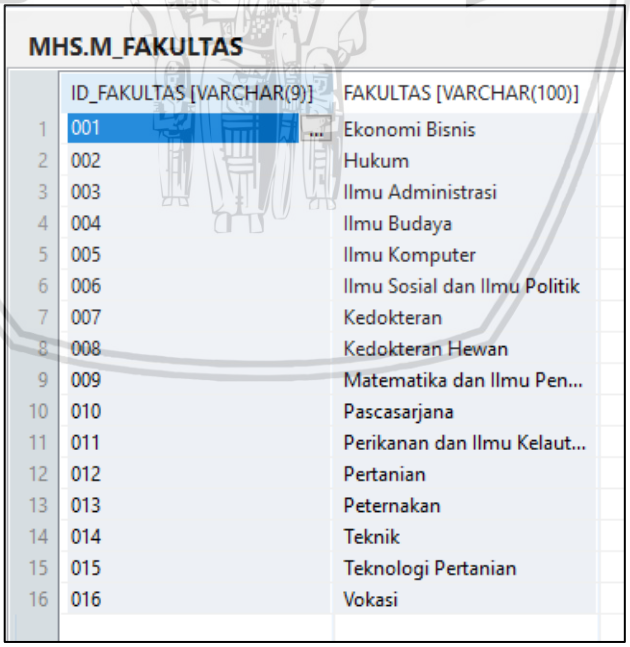
Gambar 5.3 Jendela Import Utility

4. Kemudian tunggu beberapa saat sampai ada pemberitahuan bahwa data telah berhasil di *import.* Untuk melihat bahwa data telah masuk kedalam tabel langkahnya adalah sebagai berikut: klik kanan pada tabel yang terkait > pilih *Data* > pilih *Browse Data*, seperti pada Gambar 5.4.



Gambar 5.4 Menampilkan Data Pada Tabel

5. Jika keseluruhan proses *import* berhasil maka akan dapat dilihat data tersebut sudah berada didalam tabel seperti pada Gambar 5.5.



Gambar 5.5 Data Hasil Proses Import

5.3.2 Data Dummy

Data *dummy* diperoleh dari hasil pembuatan otomatis (*generate*) oleh komputer dengan bantuan progam Microsoft Office Excel. Data-data tersebut meliputi antara lain: Data Mahasiswa, Data Biodata Mahasiswa, Data Sekolah Asal

Mahasiswa, serta Data KHS Mahasiswa. Proses pembuatan data *dummy* kurang lebihnya seperti yang dapat dilihat pada Gambar 5.6.

	A	B	C	D	E
1	**NIM**	**ID_SELEKSI**	**ID_PROG_STUDI**	**NAMA**	**ANGKATAN**
2	UB00000001	03	083	MHS00000001	2011
3	UB00000002	04	059	MHS00000002	2018
4	UB00000003	01	089	MHS00000003	2016
5	UB00000004	02	074	MHS00000004	2017
6	UB00000005	02	128	MHS00000005	2018
7	UB00000007	04	075	MHS00000007	2014
8	UB00000008	04	023	MHS00000008	2016
9	UB00000009	02	009	MHS00000009	2011
10	UB00000010	05	046	MHS00000010	2017
11	UB00000011	05	057	MHS00000011	2012
12	UB00000013	01	080	MHS00000013	2016
13	UB00000014	01	065	MHS00000014	2018
14	UB00000016	03	002	MHS00000016	2015
15	UB00000017	03	009	MHS00000017	2017
16	UB00000018	05	080	MHS00000018	2012
17	UB00000019	05	113	MHS00000019	2013
18	UB00000020	03	090	MHS00000020	2014
19	UB00000021	03	117	MHS00000021	2014
20	UB00000023	05	058	MHS00000023	2017
21	UB00000024	04	058	MHS00000024	2011

Gambar 5.6 Data Dummy Mahasiswa

Proses memasukkan data *dummy* kedalam basis data menggunakan cara yang sama dengan proses memasukkan data asli.

5.4 Implementasi Materialized Query Table

Disini akan membahas tentang mekanisme implementasi MQT dengan menerjemahkan aljabar relasional yang telah dirancang sebelumnya kedalam bentuk *Data Definition Language* (DDL) sehingga dapat di eksekusi (*deploy*) kedalam basis data.

5.4.1 MQT Sebaran Mahasiswa

MQT ini berfungsi untuk menyimpan data perhitungan sebaran mahasiswa. Dimensi perhitungan yang digunakan antara lain adalah fakultas, jurusan, jenjang, program studi, dan angkatan. Untuk itu tabel-tabel yang diperlukan antara lain adalah MAHASISWA, M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS. Adapun penulisan *query* berdasarkan algoritme aljabar relasional yang telah dirancang sebelumnya di Bab 4 pada Gambar 4.4 adalah sebagaimana ditunjukkan pada Gambar 5.7.

```
1  CREATE TABLE MQT_SEBARAN_MHS AS (
2    SELECT
3      M_FAKULTAS.FAKULTAS, M_JURUSAN.JURUSAN,
4      M_JENJANG.JENJANG, M_PROG_STUDI.PROG_STUDI,
5      MAHASISWA.ANGKATAN, COUNT(MAHASISWA.NIM) JUMLAH
6    FROM
7      MAHASISWA, M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS
8    WHERE
9      M_PROG_STUDI.ID_PROG_STUDI = MAHASISWA.ID_PROG_STUDI
10     AND M_JENJANG.ID_JENJANG = M_PROG_STUDI.ID_JENJANG
11     AND M_JURUSAN.ID_JURUSAN = M_PROG_STUDI.ID_JURUSAN
12     AND M_FAKULTAS.ID_FAKULTAS = M_JURUSAN.ID_FAKULTAS
13   GROUP BY
14     M_FAKULTAS.FAKULTAS, M_JURUSAN.JURUSAN, M_JENJANG.JENJANG,
15     M_PROG_STUDI.PROG_STUDI, MAHASISWA.ANGKATAN
16 ) DATA INITIALLY DEFERRED REFRESH DEFERRED;
```

Gambar 5.7 Implementasi Query MQT Sebaran Mahasiswa

Penjelasan dari kode DDL pada Gambar 5.7 adalah sebagai berikut:

Baris 1 : Perintah membuat MQT dengan nama MQT_SEBARAN_MHS

Baris 2-5 : Pemilihan dari kolom yang ditampilkan sekaligus menjadi atribut dari MQT_SEBARAN_MHS

Baris 5 : Perintah untuk melakukan fungsi agregasi COUNT yang berarti menghitung jumlah baris / jumlah mahasiswa

Baris 6-7 : Penggabungan tabel induk tempat rujukan data dari MQT yaitu tabel MAHASISWA, M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS

Baris 8-12 : Definisi hubungan antar tabel yang menjadi syarat penggabungan tabel-tabel induk dari MQT

Baris 13-15 : Perintah untuk melakukan pengelompokkan data (*grouping*) berdasarkan kolom FAKULTAS, JURUSAN, JENJANG, PROG_STUDI dan ANGKATAN

Baris 16 : Perintah untuk mengakhiri klausa pembuatan MQT dengan ketentuan seperti yang dijelaskan pada bab sebelumnya yaitu data tidak langsung disegarkan ketika MQT dibuat serta MQT dapat di mutakhirkan kapanpun dengan perintah REFRESH.

5.4.2 MQT Sebaran Mahasiswa berdasarkan Sekolah Asal

MQT ini berfungsi untuk menyimpan data perhitungan sebaran mahasiswa berdasarkan dari sekolah asal. Dimensi perhitungan yang digunakan antara lain adalah fakultas, jurusan, jenjang, program studi, dan sekolah asal. Untuk itu tabel-tabel yang diperlukan antara lain adalah MHS_SEKOLAH, M_SEKOLAH, MAHASISWA, M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS. Adapun penulisan *query* berdasarkan algoritme aljabar relasional yang telah dirancang

sebelumnya di Bab 4 pada Gambar 4.6 adalah sebagaimana ditunjukkan pada Gambar 5.8.

```
1  CREATE TABLE MQT_SEBARAN_SEKOLAH AS (
2    SELECT
3      M_FAKULTAS.FAKULTAS, M_JURUSAN.JURUSAN,
4      M_JENJANG.JENJANG, M_PROG_STUDI.PROG_STUDI,
5      M_SEKOLAH.SEKOLAH, COUNT(MHS_SEKOLAH.NIM) JUMLAH
6    FROM
7      MHS_SEKOLAH, M_SEKOLAH, MAHASISWA,
8      M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS
9    WHERE
10     M_SEKOLAH.ID_SEKOLAH = MHS_SEKOLAH.ID_SEKOLAH
11     AND MAHASISWA.NIM = MHS_SEKOLAH.NIM
12     AND M_PROG_STUDI.ID_PROG_STUDI = MAHASISWA.ID_PROG_STUDI
13     AND M_JENJANG.ID_JENJANG = M_PROG_STUDI.ID_JENJANG
14     AND M_JURUSAN.ID_JURUSAN = M_PROG_STUDI.ID_JURUSAN
15     AND M_FAKULTAS.ID_FAKULTAS = M_JURUSAN.ID_FAKULTAS
16   GROUP BY
17     M_FAKULTAS.FAKULTAS, M_JURUSAN.JURUSAN,
18     M_JENJANG.JENJANG, M_PROG_STUDI.PROG_STUDI,
19     M_SEKOLAH.SEKOLAH
20 ) DATA INITIALLY DEFERRED REFRESH DEFERRED;
```

Gambar 5.8 Implementasi Query MQT Sebaran Mahasiswa berdasarkan Sekolah Asal

Penjelasan dari kode DDL pada Gambar 5.8 adalah sebagai berikut:

Baris 1 : Perintah membuat MQT dengan nama MQT_SEBARAN_SEKOLAH

Baris 2-5 : Pemilihan dari kolom yang ditampilkan sekaligus menjadi atribut dari MQT_SEBARAN_SEKOLAH

Baris 5 : Perintah untuk melakukan fungsi agregasi COUNT yang berarti menghitung jumlah baris / jumlah mahasiswa

Baris 6-8 : Penggabungan tabel induk tempat rujukan data dari MQT yaitu tabel MHS_SEKOLAH, M_SEKOLAH, MAHASISWA, M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS

Baris 9-15 : Definisi hubungan antar tabel yang menjadi syarat penggabungan tabel-tabel induk dari MQT

Baris 16-19 : Perintah untuk melakukan pengelompokkan data (*grouping*) berdasarkan kolom FAKULTAS, JURUSAN, JENJANG, PROG_STUDI, SEKOLAH

Baris 20 : Perintah untuk mengakhiri klausa pembuatan MQT dengan ketentuan seperti yang dijelaskan pada bab sebelumnya yaitu data tidak langsung disegarkan ketika MQT dibuat serta MQT dapat di mutakhirkan kapanpun dengan perintah REFRESH.

5.4.3 MQT Sebaran Mahasiswa berdasarkan Daerah Asal

MQT ini berfungsi untuk menyimpan data perhitungan sebaran mahasiswa berdasarkan dari daerah asal. Dimensi perhitungan yang digunakan antara lain adalah fakultas, jurusan, jenjang, program studi, propinsi dan kota. Untuk itu tabel-tabel yang diperlukan antara lain adalah MHS_BIODATA, M_KOTA, M_PROPINSI, MAHASISWA, M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS. Adapun penulisan *query* berdasarkan algoritme aljabar relasional yang telah dirancang sebelumnya di Bab 4 pada Gambar 4.8 adalah sebagaimana ditunjukkan pada Gambar 5.9.

```
1  CREATE TABLE MQT_SEBARAN_DAERAH AS (
2    SELECT
3      M_FAKULTAS.FAKULTAS, M_JURUSAN.JURUSAN,
4      M_JENJANG.JENJANG, M_PROG_STUDI.PROG_STUDI,
5      M_PROPINSI.PROPINSI, M_KOTA.KOTA,
6      COUNT(MHS_BIODATA.NIM) JUMLAH
7    FROM
8      MHS_BIODATA, M_KOTA, M_PROPINSI, MAHASISWA,
9      M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS
10   WHERE
11     MAHASISWA.NIM = MHS_BIODATA.NIM
12     AND M_PROG_STUDI.ID_PROG_STUDI = MAHASISWA.ID_PROG_STUDI
13     AND M_JENJANG.ID_JENJANG = M_PROG_STUDI.ID_JENJANG
14     AND M_JURUSAN.ID_JURUSAN = M_PROG_STUDI.ID_JURUSAN
15     AND M_FAKULTAS.ID_FAKULTAS = M_JURUSAN.ID_FAKULTAS
16     AND M_KOTA.ID_KOTA = MHS_BIODATA.ID_KOTA
17     AND M_PROPINSI.ID_PROPINSI = M_KOTA.ID_PROPINSI
18   GROUP BY
19     M_FAKULTAS.FAKULTAS, M_JURUSAN.JURUSAN,
20     M_JENJANG.JENJANG, M_PROG_STUDI.PROG_STUDI,
21     M_PROPINSI.PROPINSI, M_KOTA.KOTA
22 ) DATA INITIALLY DEFERRED REFRESH DEFERRED;
```

Gambar 5.9 Implementasi Query MQT Sebaran Mahasiswa berdasarkan Daerah Asal

Penjelasan dari kode DDL pada Gambar 5.9 adalah sebagai berikut:

Baris 1	:	Perintah membuat MQT dengan nama MQT_SEBARAN_DAERAH
Baris 2-6	:	Pemilihan dari kolom yang ditampilkan sekaligus menjadi atribut dari MQT_SEBARAN_DAERAH
Baris 6	:	Perintah untuk melakukan fungsi agregasi COUNT yang berarti menghitung jumlah baris / jumlah mahasiswa
Baris 7-9	:	Penggabungan tabel induk tempat rujukan data dari MQT yaitu tabel MHS_BIODATA, M_KOTA, M_PROPINSI, MAHASISWA, M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS

Baris 10-18 : Definisi hubungan antar tabel yang menjadi syarat penggabungan tabel-tabel induk dari MQT

Baris 18-21 : Perintah untuk melakukan pengelompokkan data (*grouping*) berdasarkan kolom FAKULTAS, JURUSAN, JENJANG, PROG_STUDI, PROPINSI, KOTA

Baris 22 : Perintah untuk mengakhiri klausa pembuatan MQT dengan ketentuan seperti yang dijelaskan pada bab sebelumnya yaitu data tidak langsung disegarkan ketika MQT dibuat serta MQT dapat di mutakhirkan kapanpun dengan perintah REFRESH.

5.5 Implementasi Staging Table

Staging Table merupakan syarat utama agar sebuah MQT dapat di *refresh* secara *incremental*. Untuk itu disini akan membahas tentang implementasi *staging table* agar dapat dimanfaatkan oleh MQT untuk melakukan proses *incremental refresh*.

5.5.1 Staging Table Sebaran Mahasiswa

Staging Table ini berfungsi untuk menyimpan seluruh perubahan data pada tabel-tabel yang terhubung ke MQT Sebaran Mahasiswa. Pada saat dilakukan proses *refresh* dilakukan, MQT tersebut tidak akan mengakses tabel-tabel induk dimana data berada, melainkan akan mengakses *staging table* ini untuk melihat apakah telah terjadi perubahan pada tabel induk atau tidak, jika ditemukan perubahan yang terjadi maka selanjutnya data didalam MQT akan diperbarui mengikuti perubahan yang telah tercatat. Adapun klausa untuk membuat sebuah *staging table* yang terhubung dengan MQT Sebaran Mahasiswa adalah seperti ditunjukkan pada Gambar 5.10.

```
1   CREATE TABLE STG_SEBARAN_MHS FOR MQT_SEBARAN_MHS PROPAGATE IMMEDIATE;
2   SET INTEGRITY FOR STG_SEBARAN_MHS IMMEDIATE CHECKED;
```

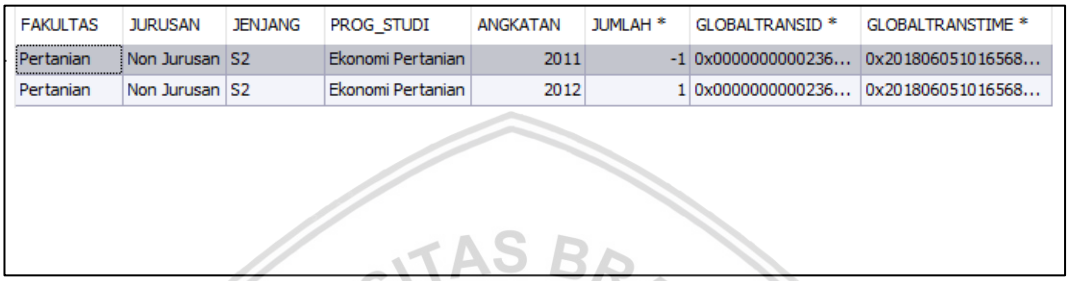
Gambar 5.10 Implementasi Query Staging Table Sebaran Mahasiswa

Penjelasan dari kode DDL pada Gambar 5.10 adalah sebagai berikut:

Baris 1 : Perintah membuat *staging table* dengan nama STG_SEBARAN_MHS yang terhubung dengan MQT_SEBARAN_MHS. Baris ini juga sekaligus memberitahukan kepada sistem bahwa semua perubahan data yang terjadi pada tabel-tabel yang terhubung dengan MQT_SEBARAN_MHS akan secara otomatis dicatat di *staging table* STG_SEBARAN_MHS.

Baris 2 : Perintah ini berfungsi untuk mengeluarkan *staging table* STG_SEBARAN_MHS dari status “pending” sehingga dapat digunakan sebagai tabel untuk mencatat perubahan data dari tabel yang terhubung dengan MQT_SEBARAN_MHS.

Setelah klausa seperti pada Gambar 5.10 berhasil dijalankan maka akan muncul sebuah tabel baru didalam *database* bernama STG_SEBARAN_MHS. Tabel ini tampak seperti tabel biasa namun kita tidak diperbolehkan untuk melakukan operasi termasuk penambahan, perubahan, ataupun penghapusan data. Namun tabel ini akan otomatis terisi ketika ada perubahan data yang terjadi pada tabel yang terhubung dengan MQT_SEBARAN_MHS. Data itulah yang nantinya akan dimanfaatkan oleh MQT untuk memperbarui data yang ada didalamnya. Lebih jelasnya seperti dapat dilihat pada Gambar 5.11.



FAKULTAS	JURUSAN	JENJANG	PROG_STUDI	ANGKATAN	JUMLAH *	GLOBALTRANSID *	GLOBALTRANSTIME *
Pertanian	Non Jurusan	S2	Ekonomi Pertanian	2011	-1	0x0000000000236...	0x201806051016568...
Pertanian	Non Jurusan	S2	Ekonomi Pertanian	2012	1	0x0000000000236...	0x201806051016568...

Gambar 5.11 Staging Table Sebaran Mahasiswa

5.5.2 Staging Table Sebaran Mahasiswa berdasarkan Sekolah Asal

Staging Table ini berfungsi untuk menyimpan seluruh perubahan data pada tabel-tabel yang terhubung ke MQT Sebaran Mahasiswa berdasarkan Sekolah Asal. Pada saat dilakukan proses *refresh* dilakukan, MQT tersebut tidak akan mengakses tabel-tabel induk dimana data berada, melainkan akan mengakses *staging table* ini untuk melihat apakah telah terjadi perubahan pada tabel induk atau tidak, jika ditemukan perubahan yang terjadi maka selanjutnya data didalam MQT akan diperbarui mengikuti perubahan yang telah tercatat. Adapun klausa untuk membuat sebuah *staging table* yang terhubung dengan MQT Sebaran Mahasiswa berdasarkan Daerah Asal adalah seperti ditunjukkan pada Gambar 5.12.

```
1  CREATE TABLE STG_SEBARAN_SEKOLAH FOR MQT_SEBARAN_SEKOLAH PROPAGATE IMMEDIATE;
2  SET INTEGRITY FOR STG_SEBARAN_SEKOLAH IMMEDIATE CHECKED;
```

Gambar 5.12 Implementasi Query Staging Table Sebaran Mahasiswa berdasarkan Sekolah Asal

Penjelasan dari kode DDL pada Gambar 5.12 adalah sebagai berikut:

Baris 1 : Perintah membuat *staging table* dengan nama STG_SEBARAN_SEKOLAH yang terhubung dengan MQT_SEBARAN_SEKOLAH. Baris ini juga sekaligus memberitahukan kepada sistem bahwa semua perubahan data yang terjadi pada tabel-tabel yang terhubung dengan MQT_SEBARAN_SEKOLAH akan secara otomatis dicatat di *staging table* STG_SEBARAN_SEKOLAH.

Baris 2 : Perintah ini berfungsi untuk mengeluarkan *staging table* STG_SEBARAN_SEKOLAH dari status "pending" sehingga dapat digunakan sebagai tabel untuk mencatat perubahan data dari tabel yang terhubung dengan MQT_SEBARAN_SEKOLAH.

Setelah klausa seperti pada Gambar 5.12 berhasil dijalankan maka akan muncul sebuah tabel baru didalam *database* bernama STG_SEBARAN_SEKOLAH. Tabel ini tampak seperti tabel biasa namun kita tidak diperbolehkan untuk melakukan operasi termasuk penambahan, perubahan, ataupun penghapusan data. Namun tabel ini akan otomatis terisi ketika ada perubahan data yang terjadi pada tabel yang terhubung dengan MQT_SEBARAN_SEKOLAH. Data itulah yang nantinya akan dimanfaatkan oleh MQT untuk memperbarui data yang ada didalamnya. Lebih jelasnya seperti dapat dilihat pada Gambar 5.13.

FAKULTAS	JURUSAN	JENJANG	PROG_STUDI	SEKOLAH	JUMLAH *	GLOBALTRANSID *	GLOBALTRANSTIME *
Ilmu Administrasi	Non Jurusan	S2	MM Pendidikan Tinggi	SEKOLAH-00001	-1	0x0000000000232...	0x201806041524128...
Ilmu Administrasi	Non Jurusan	S2	MM Pendidikan Tinggi	SEKOLAH-00002	1	0x0000000000232...	0x201806041524128...
Kedokteran	Keperawatan	S1	Ilmu Keperawatan	SEKOLAH-00003	-1	0x0000000000232...	0x201806041524128...
Kedokteran	Kedokteran	Sp-1	Patologi Klinik	SEKOLAH-00003	-1	0x0000000000232...	0x201806041524128...
Kedokteran	Keperawatan	S1	Ilmu Keperawatan	SEKOLAH-00004	1	0x0000000000232...	0x201806041524128...
Kedokteran	Kedokteran	Sp-1	Patologi Klinik	SEKOLAH-00004	1	0x0000000000232...	0x201806041524128...

Gambar 5.13 Staging Table Sebaran Mahasiswa berdasarkan Sekolah Asal

5.5.3 Staging Table Sebaran Mahasiswa berdasarkan Daerah Asal

Staging Table ini berfungsi untuk menyimpan seluruh perubahan data pada tabel-tabel yang terhubung ke MQT Sebaran Mahasiswa berdasarkan Daerah Asal. Pada saat dilakukan proses *refresh* dilakukan, MQT tersebut tidak akan mengakses tabel-tabel induk dimana data berada, melainkan akan mengakses *staging table* ini untuk melihat apakah telah terjadi perubahan pada tabel induk atau tidak, jika ditemukan perubahan yang terjadi maka selanjutnya data didalam MQT akan diperbarui mengikuti perubahan yang telah tercatat. Adapun klausa untuk membuat sebuah *staging table* yang terhubung dengan MQT Sebaran Mahasiswa berdasarkan Daerah Asal adalah seperti ditunjukkan pada Gambar 5.14.

```
1  CREATE TABLE STG_SEBARAN_DAERAH FOR MQT_SEBARAN_DAERAH PROPAGATE IMMEDIATE;
2  SET INTEGRITY FOR STG_SEBARAN_DAERAH IMMEDIATE CHECKED;
```

Gambar 5.14 Implementasi Query Staging Table Sebaran Mahasiswa berdasarkan Daerah Asal

Penjelasan dari kode DDL pada Gambar 5.14 adalah sebagai berikut:

Baris 1 : Perintah membuat *staging table* dengan nama STG_SEBARAN_DAERAH yang terhubung dengan MQT_SEBARAN_DAERAH. Baris ini juga sekaligus memberitahukan kepada sistem bahwa semua perubahan data yang terjadi pada tabel-tabel yang terhubung dengan

MQT_SEBARAN_DAERAH akan secara otomatis dicatat di *staging table* STG_SEBARAN_DAERAH.

Baris 2 : Perintah ini berfungsi untuk mengeluarkan *staging table* STG_SEBARAN_DAERAH dari status "pending" sehingga dapat digunakan sebagai tabel untuk mencatat perubahan data dari tabel yang terhubung dengan MQT_SEBARAN_DAERAH.

Setelah klausa seperti pada Gambar 5.14 berhasil dijalankan maka akan muncul sebuah tabel baru didalam *database* bernama STG_SEBARAN_DAERAH. Tabel ini tampak seperti tabel biasa namun kita tidak diperbolehkan untuk melakukan operasi termasuk penambahan, perubahan, ataupun penghapusan data. Namun tabel ini akan otomatis terisi ketika ada perubahan data yang terjadi pada tabel yang terhubung dengan MQT_SEBARAN_DAERAH. Data itulah yang nantinya akan dimanfaatkan oleh MQT untuk memperbarui data yang ada didalamnya. Lebih jelasnya seperti dapat dilihat pada Gambar 5.15.

FAKULTAS	JURUSAN	JENJANG	PROG_STUDI	SEKOLAH	JUMLAH *	GLOBALTRANSID *	GLOBALTRANSTIME *
Ilmu Administrasi	Non Jurusan	S2	MM Pendidikan Tinggi	SEKOLAH-00001	-1	0x0000000000232...	0x201806041524128...
Ilmu Administrasi	Non Jurusan	S2	MM Pendidikan Tinggi	SEKOLAH-00002	1	0x0000000000232...	0x201806041524128...
Kedokteran	Keperawatan	S1	Ilmu Keperawatan	SEKOLAH-00003	-1	0x0000000000232...	0x201806041524128...
Kedokteran	Kedokteran	Sp-1	Patologi Klinik	SEKOLAH-00003	-1	0x0000000000232...	0x201806041524128...
Kedokteran	Keperawatan	S1	Ilmu Keperawatan	SEKOLAH-00004	1	0x0000000000232...	0x201806041524128...
Kedokteran	Kedokteran	Sp-1	Patologi Klinik	SEKOLAH-00004	1	0x0000000000232...	0x201806041524128...

Gambar 5.15 Staging Table Sebaran Mahasiswa berdasarkan Daerah Asal

BAB 6 PENGUJIAN DAN ANALISIS

Bab ini akan menjelaskan mengenai hasil pengujian serta analisis dari *incremental refresh materialized query table* (MQT) memanfaatkan *staging table* untuk mengetahui bagaimana optimasi yang dihasilkan terhadap waktu eksekusi query dan sumber daya yang digunakan. Seperti yang telah dijabarkan pada bab 3 bahwa pengujian yang dilakukan adalah pengujian performa. Pengujian ini dilakukan dengan membandingkan performa dari mekanisme *full refresh* dengan *incremental refresh*.

6.1 Pengujian Performa

Pengujian performa *(performance testing)* yang dilakukan pada penelitian ini menggunakan bantuan dua indikator pengukuran antara lain kecepatan eksekusi *query* dan *resource* yang dibutuhkan dalam hal ini adalah penggunaan CPU. Kedua indikator tersebut digunakan untuk menyatakan besaran dari hasil pengukuran dan pembandingan terhadap mekanisme *full refresh* dan *incremental* refresh. Untuk memastikan bahwa *incremental refresh* dapat berjalan maka perlu dibuat skenario pengujian, yaitu dengan melakukan beberapa kali perubahan data pada tabel utama kemudian memastikan bahwa perubahan tersebut telah berhasil masuk kedalam *materialize query table*. Untuk proses pengujiannya akan menggunakan bantuan *tools* IBM Data Studio.

6.1.1 MQT Sebaran Mahasiswa

MQT ini berfungsi untuk menyimpan jumlah sebaran mahasiswa di Universitas Brawijaya. Data yang disimpan pada MQT ini antara lain adalah: fakultas, jurusan, jenjang, program studi, angkatan dan jumlah mahasiswa. Data yang akan diproses kedalam MQT ini sebanyak 1.978.853 baris dan masuk kedalam kategori data besar. Skenario dalam pengujian ini adalah dengan melakukan percobaan sebanyak 10 kali dengan rincian sebagai berikut:

Tabel 6.1 Skenario Pengujian MQT Sebaran Mahasiswa

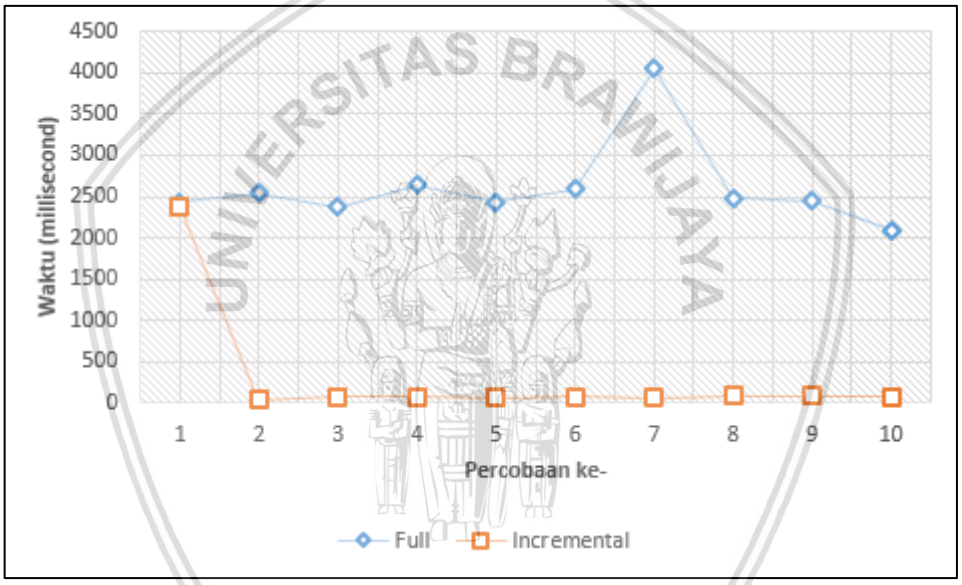
Percobaan ke-	Persentase Perubahan	Jumlah Data yang Dirubah
1	0%	Inisialisasi MQT
2	0%	0
3	10%	197.885
4	20%	395.771
5	30%	593.656
6	40%	791.541
7	50%	989.427
8	60%	1.187.312
9	70%	1.358.197
10	80%	1.583.082

Pengukuran dilakukan setelah masing-masing perubahan data dilakukan. Skenario pengujian terhadap MQT ini seperti yang dapat dilihat pada Tabel 6.1

Tabel 6.2 Hasil Perhitungan Waktu Eksekusi MQT Sebaran Mahasiswa

Jenis Refresh	Percobaan (millisecond)										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
Full	2435	2542	2372	2634	2439	2593	4054	2476	2444	2098	2608.7
Incremental	2375	37	77	72	61	77	62	90	94	73	301.8

Seperti yang dapat dilihat pada Tabel 6.2 bahwa waktu yang dibutuhkan untuk melakukan *refresh* secara *incremental* lebih cepat dibandingkan secara *full*. Lebih jelasnya akan digambarkan menggunakan grafik perbandingan seperti ditunjukkan pada Gambar 6.1



Gambar 6.1 Grafik Perbandingan Eksekusi MQT Sebaran Mahasiswa

Seperti ditunjukkan pada Gambar 6.1 dapat dilihat perbedaan waktu yang signifikan antara kedua mekanisme *refresh* tersebut. Pada percobaan pertama jumlah waktu yang dibutuhkan tidak jauh berbeda karena keduanya melakukan perhitungan awal pada semua data. Namu pada percobaan kedua dan selanjutnya proses *incremental refresh* menunjukan penurunan waktu yang signifikan. Hal ini disebabkan karena proses *full refresh* selalu menghitung ulang semua data mahasiswa sedangkan *incremental refresh* hanya menghitung sejumlah data yang mengalami perubahan saja.

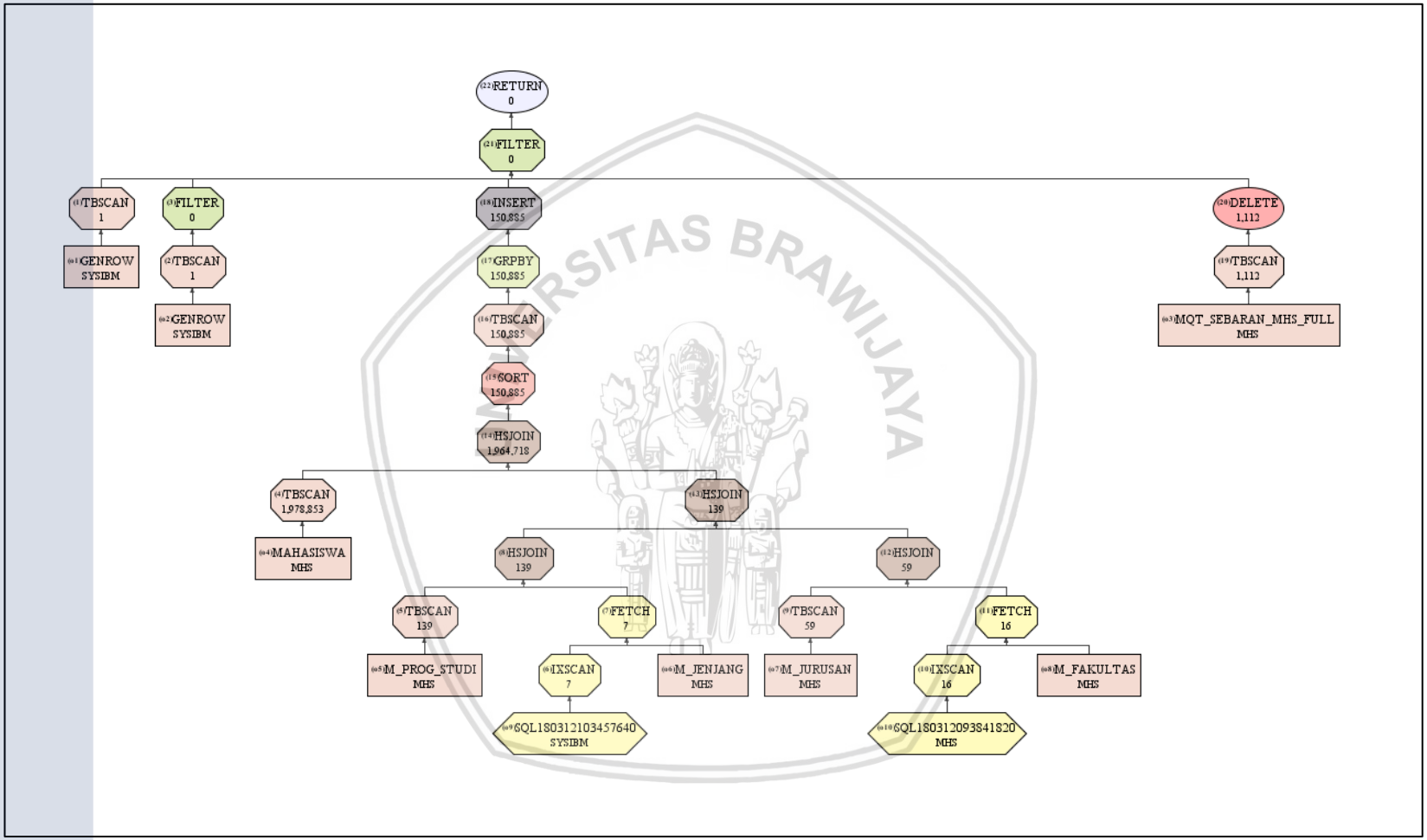
Setelah mendapatkan hasil pengukuran waktu eksekusi *query*, maka selanjutnya adalah mengukur besaran sumber daya yang dibutuhkan pada saat proses *refresh* MQT dilakukan. Besaran sumber daya diukur dari dua aspek yaitu biaya CPU yang digunakan, dan biaya *input* dan *output* (I/O) yang dibutuhkan.

Pengukuran terhadap sumber daya yang dibutuhkan menggunakan bantuan *tools* IBM Data Studio. Adapun hasil pengukurannya seperti ditunjukan pada tabel 6.3.

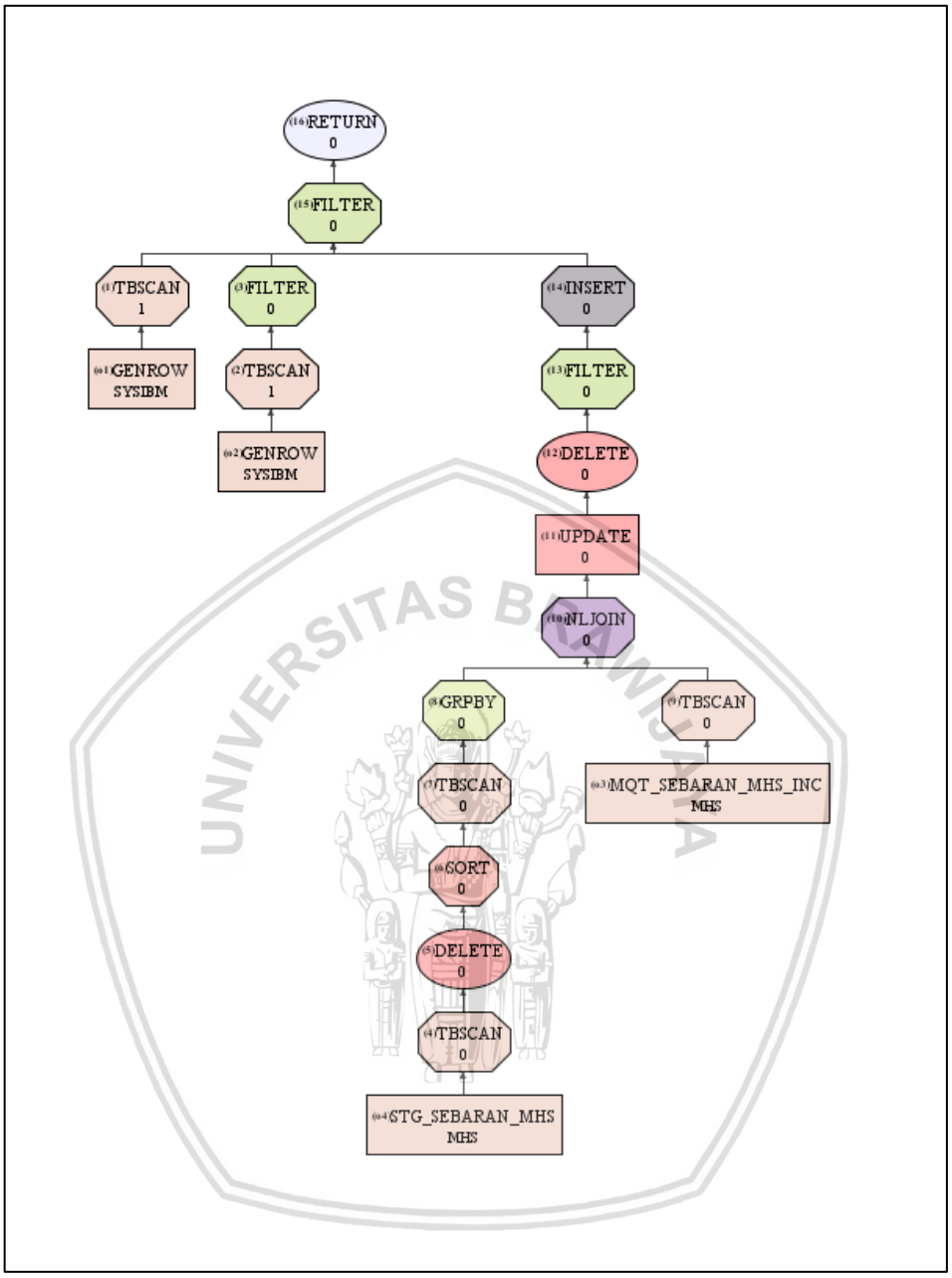
Tabel 6.3 Hasil Perhitungan Sumber Daya MQT Sebaran Mahasiswa

Jenis Refresh	Sumber Daya (*timerons*)		
	Biaya CPU	Biaya IO	Total Biaya
Full	238,619,863.50	42,421.00	84,940.72
Incremental	3,561,490.50	597.47	1286.29

Tabel 6.3 menunjukkan perbedaan yang signifikan antara biaya sumber daya yang digunakan pada saat proses *refresh* dilakukan. Sumber daya yang diperlukan oleh mekanisme *full refresh* jauh lebih besar dibandingkan dengan *incremental refresh*. Hal ini disebabkan karena *full refresh* melakukan banyak operasi pada tabel-tabel induk yang terhubung dengan MQT. Sebaliknya *incremental refresh* hanya melakukan operasi terhadap MQT itu sendiri dan *staging table* yang terhubung kepadanya. Untuk lebih detailnya dapat dianalisa dengan menggunakan diagram *access plan* seperti yang ditunjukkan pada Gambar 6.2 dan Gambar 6.3.



Gambar 6.2 Diagram Access Plan MQT Sebaran Mahasiswa dengan Full Refresh



Gambar 6.3 Diagram Access Plan MQT Sebaran Mahasiswa dengan Incremental Refresh

Seperti yang dapat dilihat pada Gambar 6.2 bahwa mekanisme *full refresh* menjalankan operasi pada banyak tabel induk seperti MAHASISWA, M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS sebelum akhirnya data yang sudah diproses akan dimasukkan kedalam MQT. Sebaliknya pada Gambar 6.3 mekanisme *incremental refresh* hanya melakukan operasi terhada MQT_SEBARAN_MHS_INC serta *staging table* yang terhubung kepadanya yaitu STG_SEBARAN_MHS. Banyaknya operasi yang harus dilakukan oleh mekanisme *full refresh* tentu akan berdampak pada besarnya sumber daya yang digunakan

sehingga biaya yang diperlukan menjadi lebih besar dibandingkan dengan mekanisme *incremental refresh*.

6.1.2 MQT Sebaran Mahasiswa berdasarkan Sekolah Asal

MQT ini berfungsi untuk menyimpan jumlah sebaran mahasiswa di Universitas Brawijaya berdasarkan sekolah asalnya. Data yang disimpan pada MQT ini antara lain adalah: fakultas, jurusan, jenjang, program studi, asal sekolah dan jumlah mahasiswa. Data yang akan diproses kedalam MQT ini sebanyak 978.853 baris baris dan masuk kedalam kategori data sedang. Skenario dalam pengujian ini adalah dengan melakukan percobaan sebanyak 10 kali dengan rincian sebagai berikut:

Tabel 6.4 Skenario Pengujian MQT Sebaran Mahasiswa berdasarkan Sekolah Asal

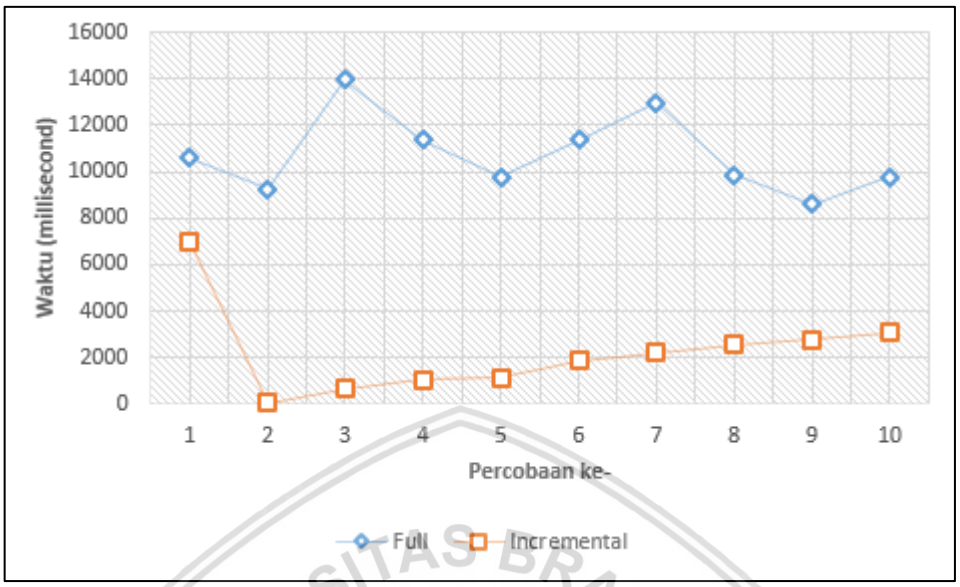
Percobaan ke-	Persentase Perubahan	Jumlah Data yang Dirubah
1	0%	Inisialisasi MQT
2	0%	0
3	10%	97.885
4	20%	195.771
5	30%	293.656
6	40%	391.541
7	50%	489.427
8	60%	587.312
9	70%	685.197
10	80%	783.082

Pengukuran dilakukan setelah masing-masing perubahan data dilakukan. Skenario pengujian terhadap MQT ini seperti yang dapat dilihat pada Tabel 6.4.

Tabel 6.5 Hasil Perhitungan Waktu Eksekusi MQT Sebaran Mahasiswa berdasarkan Sekolah Asal

Jenis Refresh	Percobaan (millisecond)										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
Full	10581	9249	13945	11391	9795	11379	12958	9871	8596	9785	10755
Incremental	6978	64	643	1046	1157	1890	2228	2572	2795	3085	2245.8

Seperti yang dapat dilihat pada Tabel 6.5 bahwa waktu yang dibutuhkan untuk melakukan *refresh* secara *incremental* lebih cepat dibandingkan secara *full*. Lebih jelasnya akan digambarkan menggunakan grafik perbandingan seperti ditunjukkan pada Gambar 6.4.



Gambar 6.4 Grafik Perbandingan Eksekusi MQT Sebaran Mahasiswa berdasarkan Sekolah Asal

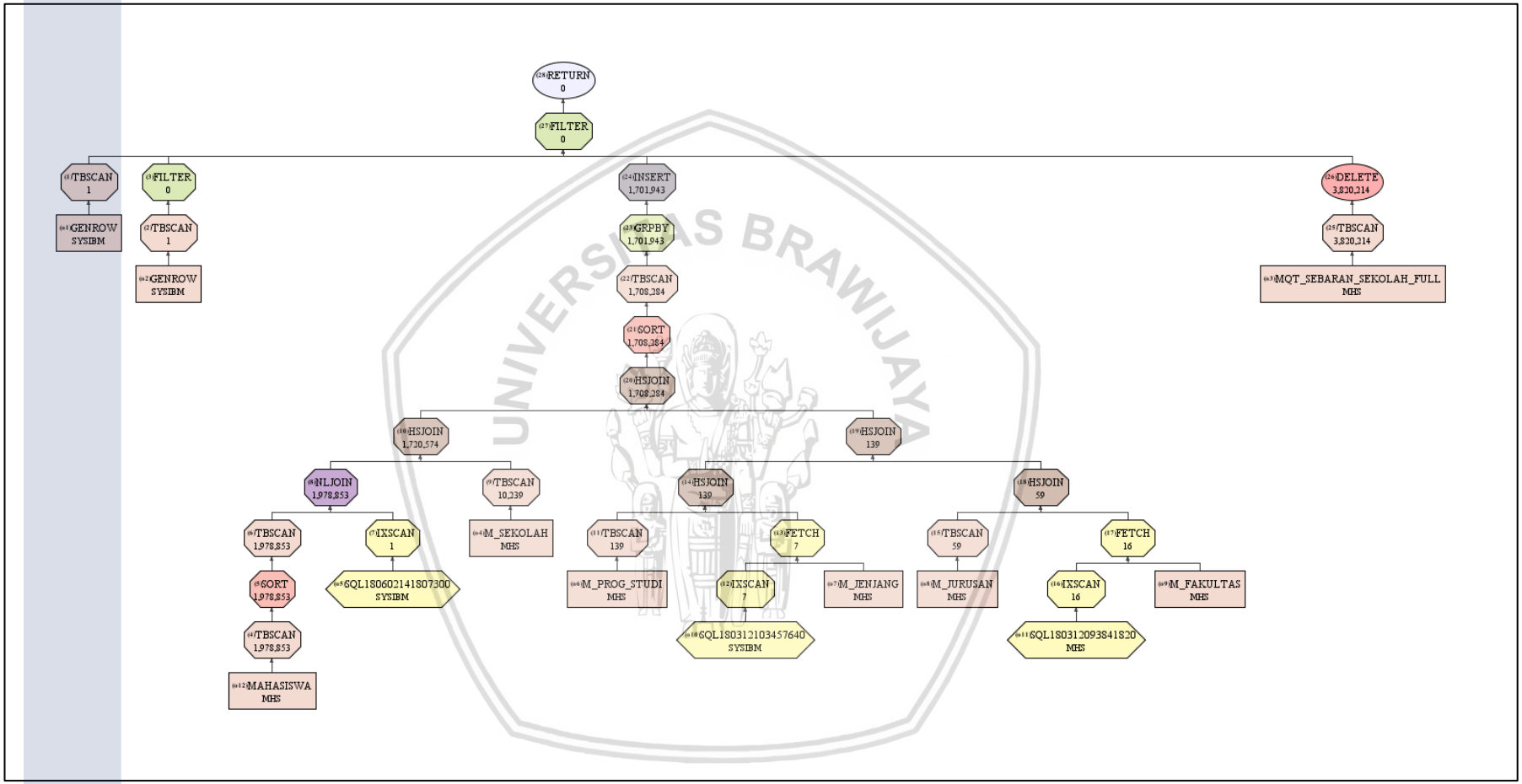
Seperti ditunjukkan pada Gambar 6.4 dapat dilihat perbedaan waktu yang signifikan antara kedua mekanisme *refresh* tersebut. Pada percobaan pertama jumlah waktu yang dibutuhkan tidak jauh berbeda karena keduanya melakukan perhitungan awal pada semua data. Namu pada percobaan kedua dan selanjutnya proses *incremental refresh* menunjukan penurunan waktu yang signifikan. Hal ini disebabkan karena proses *full refresh* selalu menghitung ulang semua data mahasiswa sedangkan *incremental refresh* hanya menghitung sejumlah data yang mengalami perubahan saja.

Setelah mendapatkan hasil pengukuran waktu eksekusi *query*, maka selanjutnya adalah mengukur besaran sumber daya yang dibutuhkan pada saat proses *refresh* MQT dilakukan. Besaran sumber daya diukur dari dua aspek yaitu biaya CPU yang digunakan, dan biaya *input* dan *output* (I/O) yang dibutuhkan. Pengukuran terhadap sumber daya yang dibutuhkan menggunakan bantuan *tools* IBM Data Studio. Adapun hasil pengukurannya seperti ditunjukan pada tabel 6.6.

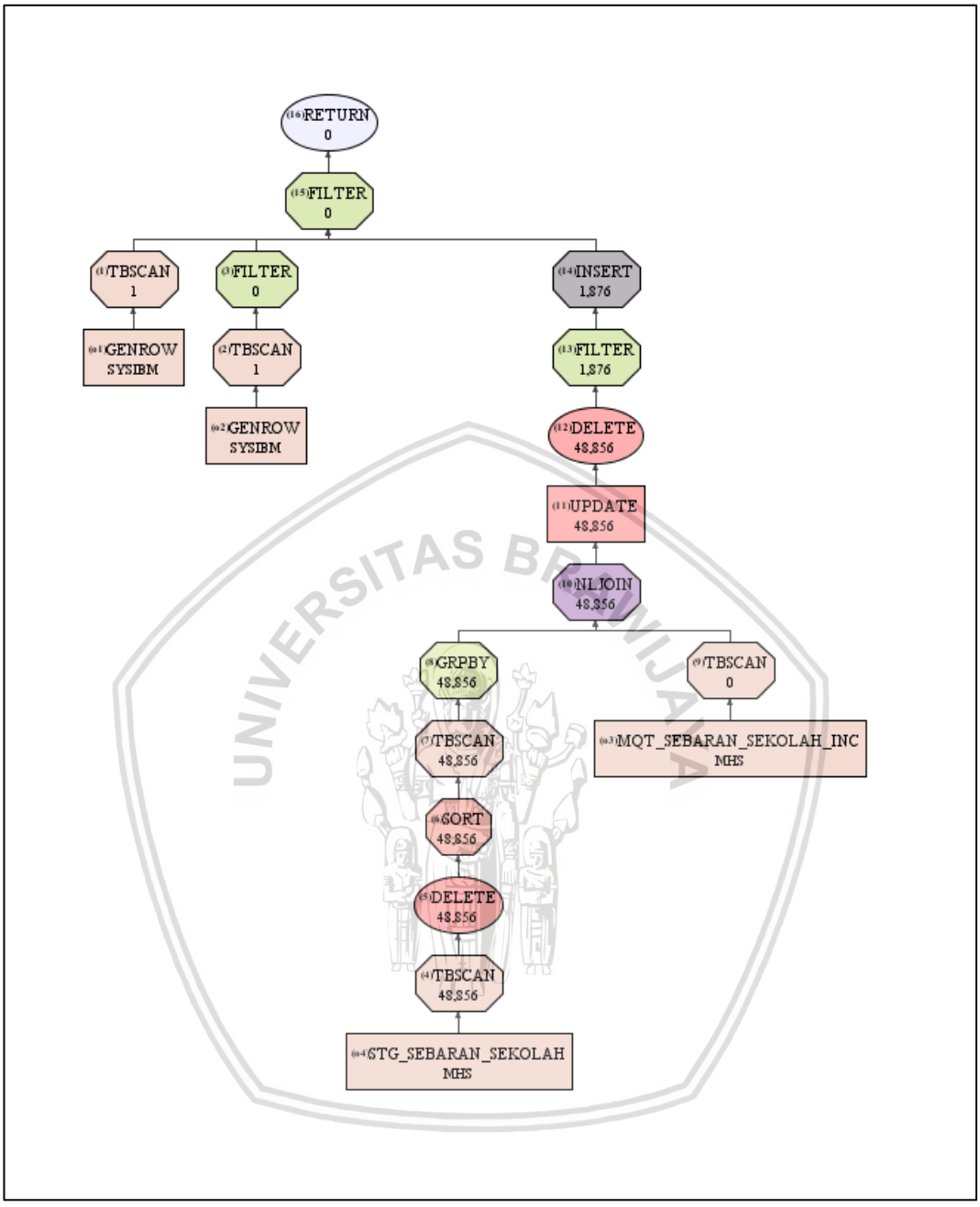
Tabel 6.6 Hasil Perhitungan Sumber Daya MQT Sebaran Mahasiswa berdasarkan Sekolah Asal

Jenis Refresh	Sumber Daya (*timerons*)		
	Biaya CPU	Biaya IO	Total Biaya
Full	143,291,645,952.00	3,967,955.50	26,225,138.00
Incremental	50,907,213,824.00	60,100.00	408,395.66

Tabel 6.6 menunjukkan perbedaan yang signifikan antara biaya sumber daya yang digunakan pada saat proses *refresh* dilakukan. Sumber daya yang diperlukan oleh mekanisme *full refresh* jauh lebih besar dibandingkan dengan *incremental refresh*. Hal ini disebabkan karena *full refresh* melakukan banyak operasi pada tabel-tabel induk yang terhubung dengan MQT. Sebaliknya *incremental refresh* hanya melakukan operasi terhadap MQT itu sendiri dan *staging table* yang terhubung kepadanya. Untuk lebih detailnya dapat dianalisa dengan menggunakan diagram *access plan* seperti yang ditunjukkan pada Gambar 6.5 dan Gambar 6.6.



Gambar 6.5 Diagram Access Plan MQT Sebaran Mahasiswa berdasarkan Sekolah Asal dengan Full Refresh



Gambar 6.6 Diagram Access Plan MQT Sebaran Mahasiswa berdasarkan Sekolah Asal dengan Incremental Refresh

Seperti yang dapat dilihat pada Gambar 6.5 bahwa mekanisme *full refresh* menjalankan operasi pada banyak tabel induk seperti MHS_SEKOLAH, M_SEKOLAH, MAHASISWA, M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS sebelum akhirnya data yang sudah diproses akan dimasukkan kedalam MQT. Sebaliknya pada Gambar 6.6 mekanisme *incremental refresh* hanya melakukan operasi terhada MQT_SEBARAN_SEKOLAH_INC serta *staging table* yang terhubung kepadanya yaitu STG_SEBARAN_SEKOLAH. Banyaknya operasi yang harus dilakukan oleh mekanisme *full refresh* tentu akan berdampak pada

besarnya sumber daya yang digunakan sehingga biaya yang diperlukan menjadi lebih besar dibandingkan dengan mekanisme *incremental refresh*.

6.1.3 MQT Sebaran Mahasiswa berdasarkan Daerah Asal

MQT ini berfungsi untuk menyimpan jumlah sebaran mahasiswa di Universitas Brawijaya. Data yang disimpan pada MQT ini antara lain adalah: fakultas, jurusan, jenjang, program studi, propinsi, kota dan jumlah mahasiswa. Data yang akan diproses kedalam MQT ini sebanyak 478.853 baris baris dan masuk kedalam kategori data kecil. Skenario dalam pengujian ini adalah dengan melakukan percobaan sebanyak 10 kali dengan rincian sebagai berikut:

Tabel 6.7 Skenario Pengujian MQT Sebaran Mahasiswa berdasarkan Daerah Asal

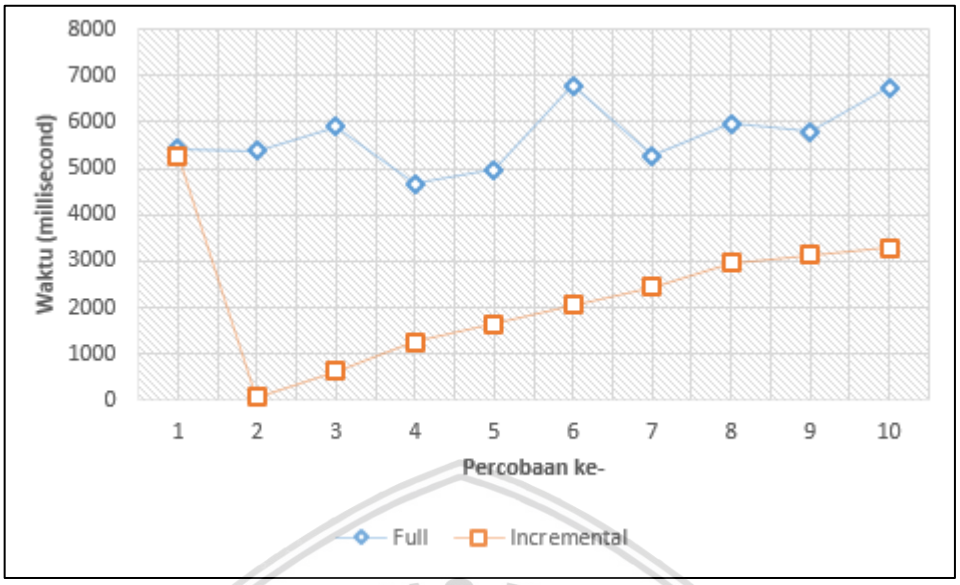
Percobaan ke-	Persentase Perubahan	Jumlah Data yang Dirubah
1	0%	Inisialisasi MQT
2	0%	0
3	10%	47.885
4	20%	95.771
5	30%	143.656
6	40%	191.541
7	50%	239.427
8	60%	287.312
9	70%	335.197
10	80%	383.082

Pengukuran dilakukan setelah masing-masing perubahan data dilakukan. Skenario pengujian terhadap MQT ini seperti yang dapat dilihat pada Tabel 6.7.

Tabel 6.8 Hasil Perhitungan Waktu Eksekusi MQT Sebaran Mahasiswa berdasarkan Daerah Asal

Jenis Refresh	Percobaan (millisecond)										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
Full	5423	5386	5905	4670	4977	6779	5280	5976	5797	6743	5693.6
Incremental	5284	76	644	1256	1638	2059	2444	2955	3132	3299	2278.7

Seperti yang dapat dilihat pada Tabel 6.8 bahwa waktu yang dibutuhkan untuk melakukan *refresh* secara *incremental* lebih cepat dibandingkan secara *full*. Lebih jelasnya akan digambarkan menggunakan grafik perbandingan seperti ditunjukkan pada Gambar 6.7



Gambar 6.7 Grafik Perbandingan Eksekusi MQT Sebaran Mahasiswa berdasarkan Daerah Asal

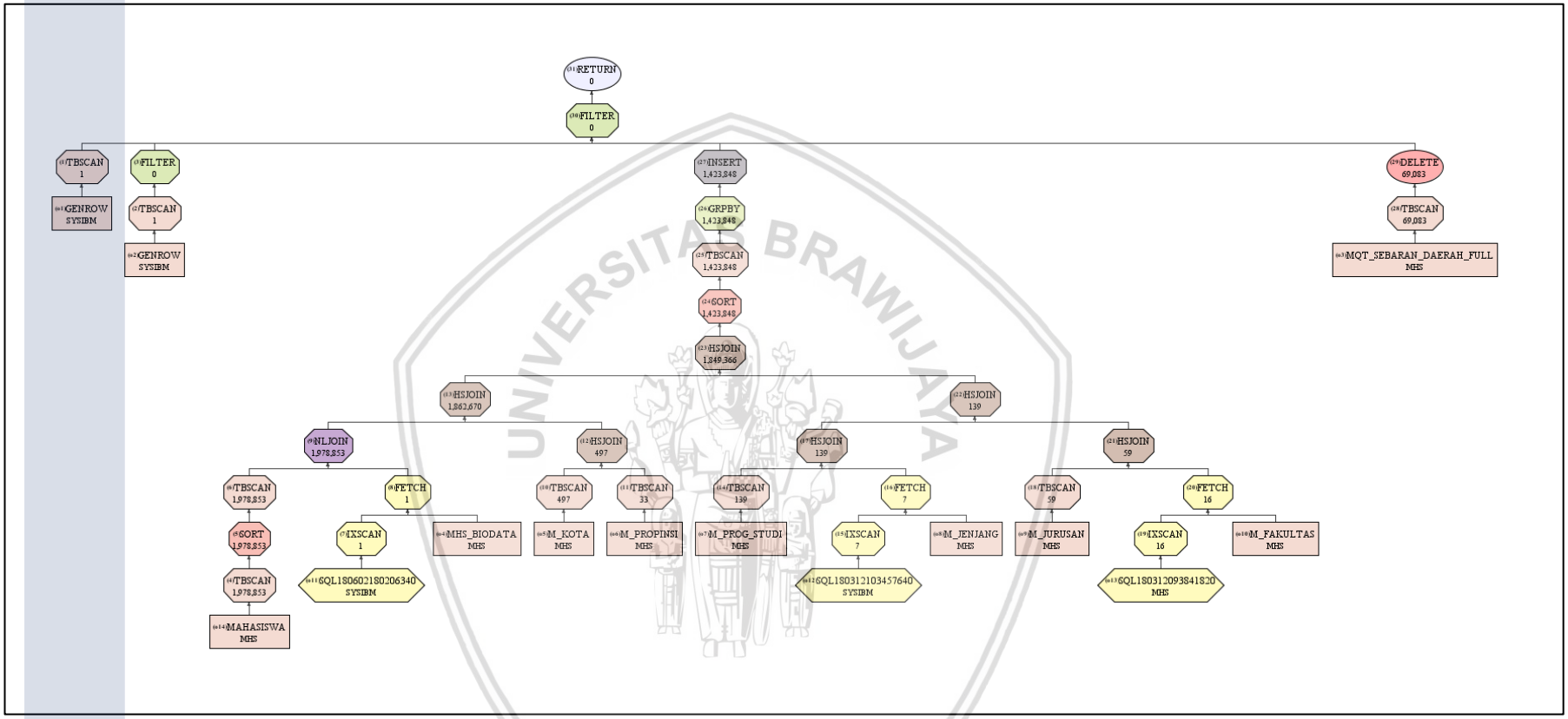
Seperti ditunjukkan pada Gambar 6.7 dapat dilihat perbedaan waktu yang signifikan antara kedua mekanisme *refresh* tersebut. Pada percobaan pertama jumlah waktu yang dibutuhkan tidak jauh berbeda karena keduanya melakukan perhitungan awal pada semua data. Namu pada percobaan kedua dan selanjutnya proses *incremental refresh* menunjukan penurunan waktu yang signifikan. Hal ini disebabkan karena proses *full refresh* selalu menghitung ulang semua data mahasiswa sedangkan *incremental refresh* hanya menghitung sejumlah data yang mengalami perubahan saja.

Setelah mendapatkan hasil pengukuran waktu eksekusi *query*, maka selanjutnya adalah mengukur besaran sumber daya yang dibutuhkan pada saat proses *refresh* MQT dilakukan. Besaran sumber daya diukur dari dua aspek yaitu biaya CPU yang digunakan, dan biaya *input* dan *output* (I/O) yang dibutuhkan. Pengukuran terhadap sumber daya yang dibutuhkan menggunakan bantuan *tools* IBM Data Studio. Adapun hasil pengukurannya seperti ditunjukan pada tabel 6.9.

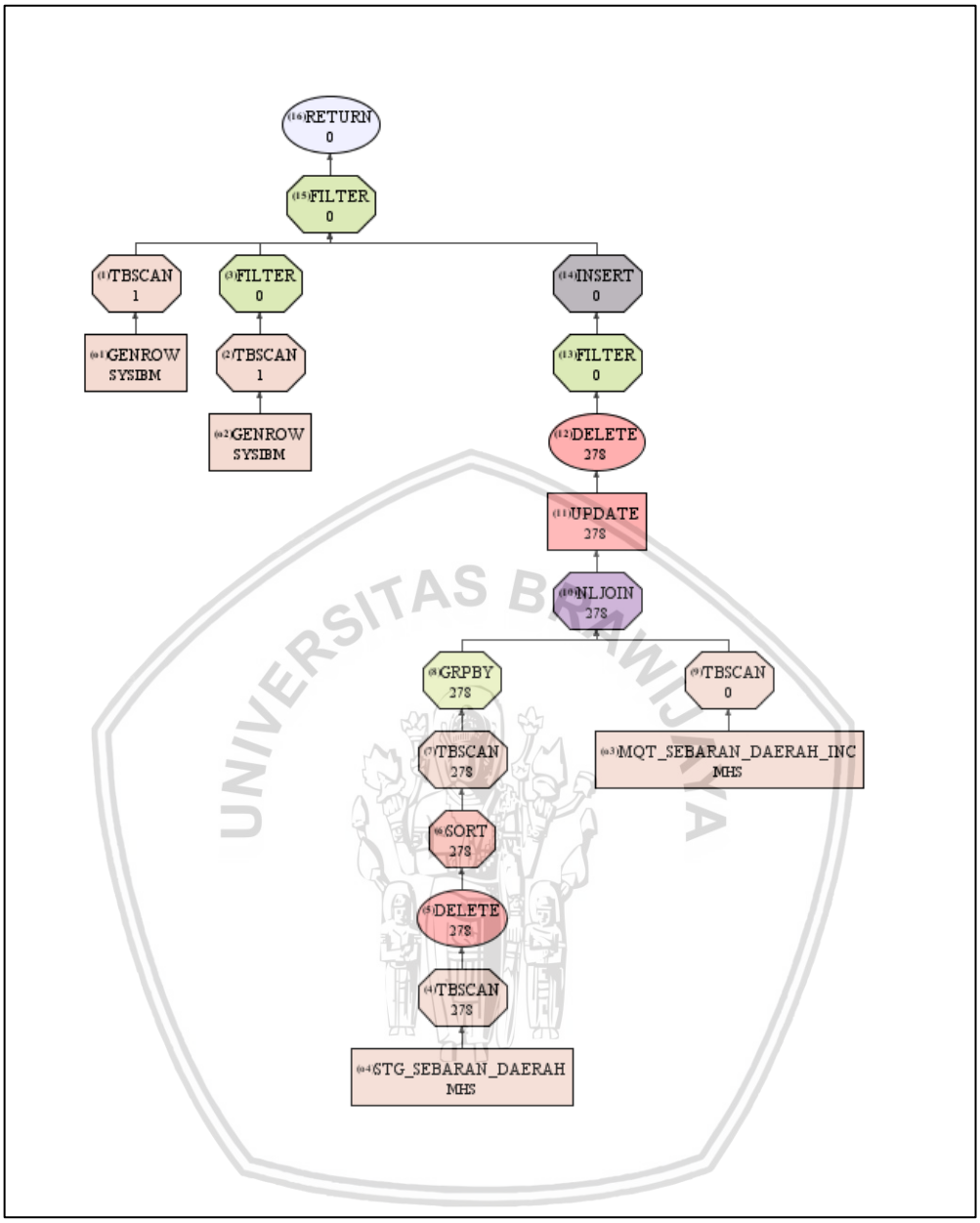
Tabel 6.9 Hasil Perhitungan Sumber Daya MQT Sebaran Mahasiswa berdasarkan Daerah Asal

Jenis Refresh	Sumber Daya (*timerons*)		
	Biaya CPU	Biaya IO	Total Biaya
Full	111,891,611,648.00	245,926.09	951,957.50
Incremental	48,418,177,024.00	4,490.00	14,014.06

Tabel 6.9 menunjukkan perbedaan yang signifikan antara biaya sumber daya yang digunakan pada saat proses *refresh* dilakukan. Sumber daya yang diperlukan oleh mekanisme *full refresh* jauh lebih besar dibandingkan dengan *incremental refresh*. Hal ini disebabkan karena *full refresh* melakukan banyak operasi pada tabel-tabel induk yang terhubung dengan MQT. Sebaliknya *incremental refresh* hanya melakukan operasi terhadap MQT itu sendiri dan *staging table* yang terhubung kepadanya. Untuk lebih detailnya dapat dianalisa dengan menggunakan diagram *access plan* seperti yang ditunjukkan pada Gambar 6.8 dan Gambar 6.9.



Gambar 6.8 Diagram Access Plan MQT Sebaran Mahasiswa berdasarkan Daerah Asal dengan Full Refresh



Gambar 6.9 Diagram Access Plan MQT Sebaran Mahasiswa berdasarkan Daerah Asal dengan Incremental Refresh

Seperti yang dapat dilihat pada Gambar 6.8 bahwa mekanisme *full refresh* menjalankan operasi pada banyak tabel induk seperti M_PROG_STUDI, M_JENJANG, M_JURUSAN, M_FAKULTAS sebelum akhirnya data yang sudah diproses akan dimasukkan kedalam MQT. Sebaliknya pada Gambar 6.9 mekanisme *incremental refresh* hanya melakukan operasi terhada MQT_SEBARAN_MHS_INC serta *staging table* yang terhubung kepadanya yaitu STG_SEBARAN_MHS. Banyaknya operasi yang harus dilakukan oleh mekanisme *full refresh* tentu akan berdampak pada besarnya sumber daya yang digunakan sehingga biaya yang diperlukan menjadi lebih besar dibandingkan dengan mekanisme *incremental refresh*.

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan seluruh pemaparan dari bab-bab sebelumnya yang bertujuan untuk memodelkan dan mengimplementasikan metode *incremental refresh* memanfaatkan *staging table* serta membandingkannya dengan metode *full refresh* pada *materialized query table* untuk optimasi *query execution time* dan *resource* yang digunakan, maka dapat ditarik kesimpulan dari penelitian ini:

1. Sebuah *Materialized Query Table (MQT)* dengan mekanisme *full refresh* terhubung secara langsung ke tabel utama. Hal ini dikarenakan pada saat *refresh* berlangsung, MQT mengambil dan mengkalkulasi ulang seluruh data dari tabel utama. Sedangkan MQT dengan mekanisme *incremental refresh* tidak terhubung secara langsung ke tabel utama, melainkan melalui sebuah *staging table* yang berfungsi untuk mencatat seluruh perubahan data pada tabel utama. Pada saat *incremental refresh* berjalan, MQT tidak mengkalkulasi data pada tabel utama, melainkan hanya memproses data yang ada didalam *staging table*.
2. Implementasi *incremental refresh* dengan memanfaatkan *staging table* pada *materialized query table* mampu meningkatkan performa *database* secara signifikan dibandingkan dengan *full refresh*. Pada eksekusi pertama tidak terdapat perbedaan yang jauh karena kedua mekanisme baik *incremental* maupun *full* sama-sama melakukan inisialisasi data kedalam MQT. Namun pada eksekusi kedua dan selanjutnya, *incremental refresh* menunjukkan penurunan waktu eksekusi yang signifikan dengan rata-rata 301.8 *milisecond* pada kategori data besar, 2245.8 *milisecond* pada kategori data sedang, dan 2278.7 pada kategori data kecil. Sedangkan untuk *full refresh* memiliki waktu eksekusi *query* dengan rata-rata 2608.7 *milisecond* pada kategori data besar, 10755 *milisecond* pada kategori data sedang, dan 5693.6 pada kategori data kecil. Secara umum dapat disimpulkan bahwa *incremental refresh* mampu meningkatkan performa database hingga mencapai 10 kali lipat dibandingkan dengan *full refresh*.
3. Implementasi *incremental refresh* dengan memanfaatkan *staging table* pada *materialized query table* mampu meningkatkan efisiensi penggunaan *resources* dibandingkan dengan *full refresh*. *Incremental refresh* menggunakan total sumberdaya sebanyak 1286.29 *timerons* pada kategori data besar, 408395.66 *timerons* pada kategori data sedang, dan 14014.06 *timerons* pada kategori data kecil. Sedangkan *full refresh* menggunakan sumberdaya sebanyak 84,940.72 *timerons* pada kategori data besar, 26,25,138 *timerons* pada kategori data sedang, dan 951,957.50 pada kategori data kecil. Secara umum dapat disimpulkan bahwa *incremental refresh* mampu meningkatkan performa *database* hingga mencapai lebih dari 50 kali lipat dibandingkan dengan *full refresh*.

7.2 Saran

Dari hasil pemaparan penelitian ini, ada beberapa saran yang ingin penulis sampaikan diantaranya:

1. Untuk lebih mengoptimalkan proses *refresh* MQT maka disarankan untuk lebih memperhatikan *query* yang digunakan dalam membangun MQT karena dapat berdampak pada kecepatan *refresh* MQT itu sendiri.
2. Disarakan untuk lebih memperhatikan penggunaan *index column* sebagai dimensi perhitungan MQT karena hal itu dapat mempengaruhi kecepatan pemrosesan *query*.

DAFTAR PUSTAKA

Broughton, C. M., 2005. *IBM DB2 Cube Views and DB2 Materialized Query Tables in a SAS® Environment*.

Gupta, A. et al., 2001. *Adapting materialized views after redefinitions: techniques and*. *Information Systems.*

Gosain, A. et al., 2015. *Architecture Based Materialized View Evolution: A Review.* ICCC.

Sharma, B. et al., 2014. *Performance Tuning in Database Management System based on Analysis of Combination of Time and Cost Parameter through Neural Network Learning*.

Haerder, T. and Reuter, A., 1983. *Principles of transaction-oriented database recovery*. *ACM Comput. Surv.*, 15(4):287–317.

Nugroho, 2015. Implementasi *Materialized Query Table* Untuk Pembangunan *Data Mart* (Studi Kasus: Universitas Brawijaya Bagian Akademik). JPTIIK.

Elmasri, R. and Navathe, S. B., 2004. *Fundamentals of Database Systems*. Addison-Wesley, 4th edition.

Lane, P. & Potineni, P., 2014. *Oracle Database Data Warehousing Guide.*

Lehner, W. et al., 2010. *MAINTENANCE OF AUTOMATIC SUMMARY TABLES IN IBM DB2/UDB,* Erlangen: IBM Almaden Research Center.

Melnyk, R., 2005. *DB2 Information Development, IBM Canada Ltd.*. [Online] Available at: https://www.ibm.com/developerworks/data/library/techarticle/dm-509melnyk/ [Diakses 1 10 2015].

Paraboschi, S., Sindoni, G., Baralis, E. & Teniente, E., 2003. *Materialized View in Multidimensional Databases. Dalam: M. Rafanelli,* penyunt. Multidimensional Databases: Problems and Solutions. Hershey: Idea Group Publishing, pp. 222-250.

Satish, S. K., Saraswatipura, M. K. & Shastry, S. C., 2007. *DB2 performance enhancements using Materialized Query Table for LUW*. *Second International Conference on Systems (ICONS'07).*

Sharma, N. et al., 2010. *Database Fundamentals.* Warden Avenue.

Silberschatz, A., Korth, H. & Sudarshan, 2009. *Database System Concepts.* 6th penyunt. New York: Mc Graw-Hill.

Steve, Hilker., 2013. *Creating an MQT Staging Table* [Online] Available at: https://community.toadworld.com/platforms/ibmdb2/w/wiki/7403.creating-an-mqt-staging-table [Diakses 15 03 2018].